3. Z39.50 Specifications for the CIP

3.1 Introduction

Z39.50 provides a general framework for the search and retrieval of data. This framework needs to be customised for each specific domain of application. This customisation is achieved here by the definition of a Z39.50 specification for the CIP in the form of a CIP profile.

The CIP profile provides the following information:

- Identification of the Z39.50 version used by the CIP (see Section 3.2).
- Identification of the objects required by the CIP (see Section 3.3).
- Identification of the communication service used by the CIP (see Section 3.4).
- Identification of the Z39.50 facilities used by the CIP and explanation about how those facilities shall be used (see Section 3.5).
- Definition of the CIP *request identifiers* (see Section 3.6).
- Management of concurrent operations by the CIP (see Section 3.7).
- Handling of diagnostic messages by the CIP (see Section 3.8).
- An explanation of the CIP specific features that have been layered onto the basic Z39.50 protocol (see Section 3.9).

The profile also defines the CIP information specific to the Earth observation domain, which shall be handled by Z39.50. This is introduced in Sections 3.5.2.2 and 3.5.3.2.2 and specified in detail in Appendices A, B and C.

3.2 Version

The CIP requires support for Z39.50-1995, referred to as Z39.50-Version 3.Whilst not all facilities of version 3 are required for the CIP, the capabilities of version 2 are not sufficient for the CIP requirements[1].

There are further specifications required to those defined in version 3, these fall into the categories of CIP object identifiers and extended services.

Whilst the CIP is a Z39.50 version 3 profile, the CIP is compatible with ANSI/NISO Z39.50-1992, referred to as Z39.50 Version 2, in order to provide some level of interoperability with profiles based on Z39.50 version 2[2].

3.3 CIP Objects

The following object identifier (OID) is assigned to the Z39.50 Version 3 standard:

{iso (1) member-body (2) US (840) ANSI-standard-Z39.50 (10003)}

This OID is abbreviated as:

[1] The subsequent sections of this document shall only define the services and objects required for the CIP Release B.

[2] Compatibility with Z39.50 Version 2 will allow the CIP to exchange messages with systems based on Z39.50 Version 2 profiles. However, for exchanges between CIP compliant systems, Z39.50 Version 3 applies.

ANSI-standard-Z39.50

Several object classes that are required by the CIP are assigned at the level immediately subordinate to the **ANSI-standard-Z39.50**, including:

- 2 = *application protocol data units* (*APDU*s) definition;
- 3 = *attribute set* definition;
- 4 = *diagnostic* definitions;
- 5 = *records syntax* definitions;
- 7 = *resource report* definitions;
- 8 = *access control* definitions;
- 9 = *extended service* definitions;
- 11 = *element specification* definitions;
- 13 = *database schema* definitions;
- 14 = *tag set* definitions.

The Z39.50 registered objects required by the CIP are presented in Table 3-1. For each object, the name of the object and its object identifier (OID) is provided. Additionally, an abbreviated name (alias), which may be used as a short name in other sections of this document, and a reference to the section in [Z3950] where the object is fully defined is also provided.

Table 3-1: Z39.50 OIDs used by the CIP

Name	Object Identifier (OID)	Alias	[Z3950] Section
Z39.50 APDUs	**{ANSI-standard-Z39.50 2 1}**	{Z39.50- APDU}	Section 4
bib-1 attribute set	**{ANSI-standard-Z39.50 3 1}**	{Z39.50-AttributeSet-bib-1}	Appendix 3, ATR.1
exp-1 attribute set	**{ANSI-standard-Z39.50 3 2}**	{Z39.50-AttributeSet-exp-1}	Appendix 3, ATR.2
ext-1 attribute set	**{ANSI-standard-Z39.50 3 3}**	{Z39.50-AttributeSet-ext-1}	Appendix 3, ATR.3
bib-1 diagnostic set	**{ANSI-standard-Z39.50 4 1}**	{Z39.50-Diagnostic-bib-1}	Appendix 4, ERR.1
diag-1 diagnostic format	**{ANSI-standard-Z39.50 4 2}**	{Z39.50-Diagnostic-diag-1}	Appendix 4, ERR.2
Explain record syntax	**{ANSI-standard-Z39.50 5 100}**	{Z39.50-RecordSyntax-Explain}	Appendix 5, REC.1
SUTRS record syntax	**{ANSI-standard-Z39.50 5 101}**	{Z39.50-RecordSyntax-SUTRS}	Appendix 5, REC.2
GRS-1 record syntax	**{ANSI-standard-Z39.50 5 105}**	{Z39.50-RecordSyntax-GRS-1}	Appendix 5, REC.5
ES Task Package record syntax	**{ANSI-standard-Z39.50 5 106}**	{Z39.50-RecordSyntax-ESTP}	Appendix 5, REC.6
Fragment record syntax	**{ANSI-standard-Z39.50 5 107}**	{Z39.50-RecordSyntax-Fragment}	Z39.50 Maintenance Agency [ZSYNT]
resource-1 resource report format	**{ANSI-standard-Z39.50 7 1}**	{Z39.50-ResourceReport-resource-1}	Appendix 6, RSC 1
Prompt access control	**{ANSI-standard-Z39.50 8 1}**	{Z39.50-AccessControl-prompt-1}	Appendix 7, ACC
Persistent Result Set extended service	**{ANSI-standard-Z39.50 9 1}**	{Z39.50-ExtendedService-PRS}	Appendix 8, EXT 1.1
Persistent Query extended service	**{ANSI-standard-Z39.50 9 2}**	{Z39.50-ExtendedService-PQ}	Appendix 8, EXT 1.2
Periodic Query Schedule extended service	**{ANSI-standard-Z39.50 9 3}**	{Z39.50-ExtendedService-PQS}	Appendix 8, EXT 1.3
Database Update extended service	**{ANSI-standard-Z39.50 9 5}**	{Z39.50-ExtendedService-DU}	Appendix 8, EXT 1.5
Search result user information format	**{ANSI-standard-Z39.50 10 1}**	{Z39.50-UserInfo-searchResult-1}	Appendix 9, USR.1
Element specification format	**{ANSI-standard-Z39.50 11 1}**	{Z39.50-ElementSpec-eSpec-1}	Appendix 10, ESP

A number of objects that are defined within this specification shall use OIDs as specified in 'OID.6 Locally Registered Objects' of Z39.50, and shall have OID of the form:

{ANSI-standard-Z39.50 n 1000 p m}

where: n = one of those numbers listed above;

p = 99 the OID index of the CIP maintenance agency as a Z39.50 registered implementor;

m = number of the object being defined.

These CIP registered objects are presented in Table 3-2. For each object, the name of the object and object identifier (OID) is provided. Additionally, an abbreviated name (alias), used as a short name in other sections of this document, and a reference to the section or appendix where the object is fully defined is provided.

Table 3-2: CIP OIDs

Name	Object Identifier (OID)	Alias	Reference
CIP Releasee B APDU	**{ANSI-standard-Z39.50 2 1000 99 1}**	{Z39.50-CIP-B-APDU}	Appendix E
CIP attribute set	**{ANSI-standard-Z39.50 3 1000 99 1}**	{Z39.50-CIP-AttributeSet}	Appendix A
CIP diagnostic set	**{ANSI-standard-Z39.50 4 1000 99 1}**	{Z39.50-CIP-Diagnostic-Set}	Appendix F
CIP order extended service	**{ANSI-standard-Z39.50 9 1000 99 1}**	{Z39.50-CIP-Order-ES}	Section 3.5.8.8
CIP collection schema	**{ANSI-standard-Z39.50 13 1000 99 1}**	{Z39.50-CIP-Schema-Collection}	Appendix C
CIP product schema	**{ANSI-standard-Z39.50 13 1000 99 2}**	{Z39.50-CIP-Schema-Product}	Appendix C
CIP user schema	**{ANSI-standard-Z39.50 13 1000 99 4}**	{Z39.50-CIP-Schema-User}	Appendix C
CIP tag set	**{ANSI-standard-Z39.50 14 1000 99 1}**	{Z39.50-CIP-TagSet}	Appendix B

3.4 Communication Services

CIP is independent of lower layer communication protocols, but it is anticipated that most implementations will be over TCP/IP. The Interoperable Catalogue System Design Document[SDD] describes the use of CIP over TCP/IP.

3.5 Z39.50 Facilities

This section presents the Z39.50 facilities and shows their use in the CIP.

Each formal issue of this document will equate to a release of the CIP, so that an implementor can develop to a defined baseline. This issue of this document (version 2.x)[3] therefore contains a detailed description of the *facilities* supported by the CIP Release B[4].

The CIP profile identifies the parts of the full Z39.50 specification[Z3950] that are applicable for the CIP Release B, thus identifying the subset of the Z39.50 *facilities* that are used for the CIP and explaining their usage for CIP purposes. Each Z39.50 *facility* is composed of one or more *services*, so the following sections contain a description of the *services* within the *facilities* that are used by the CIP. Within these *services*, the CIP identifies the optional parts of Z39.50 that are mandatory for the CIP and the optional parts of Z39.50 that are excluded from the CIP.

[3] Version 3.x will contain the description for CIP Release C.

[4] For this reason, the terms 'CIP' and 'CIP Release B' are used interchangeably in this section.

The profiling of Z39.50 for CIP purposes is achieved by including the formal Z39.50 ASN.1 definitions and customising their use by:

- striking through the 'OPTIONAL' keyword in the optional parts of the Z39.50 ASN.1 definitions, which means that the definitions are mandatory for CIP compliance;
- greying out the optional parts of the Z39.50 ASN.1 definitions, which means that the definitions are not needed for CIP compliance.
- explaining the meaning of the Z39.50 ASN.1 definitions and clarifying their use for the CIP.

The Z39.50 *facilities* used by the CIP are summarised in Table 3-3[5]. The usage of these *facilities* by the CIP is described in the following subsections.

Table 3-3: Z39.50 Facilities usage

Z39.50 Facility	**CIP**
Initialisation Facility	O
Search Facility	O
Retrieval Facility	O
Result-set-delete Facility	O
Access Control Facility	O
Accounting/Resource Control Facility	O
Sort Facility	X
Browse Facility (Scan service)	X
Extended Services Facility *Persistent Result Set* *Persistent Query* *Periodic Query Schedule* *Item Order* *Database Update* *Export Specification* *Export Invocation*	O O O O X O X X
Explain Facility	O
Termination Facility	O

[5] "O" indicates that the *facility* is supported by the CIP, whereas "X" indicates that the *facility* is not supported by the CIP.

3.5.1 Initialisation Facility

The *Init service* allows an *origin* to establish a *Z-association* with the *target*.

The CIP uses the standard Z39.59 *Init service* negotiation procedures to control the use of all *services*.

CIP Release B uses the **`idAuthentication`** parameter to support the authentication of a CIP user. A **`NULL`** (empty) value for this parameter will allow a user to establish an association with "guest" capabilities. A CIP Release B client will use this parameter to provide information to the local Retrieval Manager (*target*) to identify the user and hence allow the local Retrieval Manager to identify the group(s) that the user is a member of. These group(s) will have defined capabilities for accessing data and *services* at the local Retrieval Manager. The **`idAuthentication`** (user id) will be checked by the local Retrieval Manager when an authenticated session is requested or when the user requests an *operation* which the local Retrieval Manager wants to have authenticated. A client can request that a session be authenticated by setting the **`accessCtrl`** options in the **`InitializeRequest`** message. This will prompt the local Retrieval Manager to challenge the client identify using the *AccessControl service*. The *facility* is also used by the local Retrieval Manager to establish a session with a remote Retrieval Manager on behalf of a user.

If a non-CIP *origin* suggests the use of Z39.50 version 2 for the *Z-association* with the *target* in the *InitializeRequest*, the *target* will accept the *Z-association* with reduced functionality and make the following assumptions:

- The *Term* in the *RPNQuery* contained in a *Search request* will always be of type *general* (i.e. an *OCTET STRING*) (see also Section 3.5.2.6).
- An *Operand* in the *RPNQuery* contained in a *Search request* will contain a single *Use attribute* in the *AttributeList* (see also Section 3.5.2.6).
- The *records* are always returned according to the "Summary" *element set* and following the *SUTRS record syntax* (see also Section 3.5.3.5).

Table 3-4: Initialize Services

ASN.1 Definition	Meaning
InitializeRequest ::= SEQUENCE{ referenceId ReferenceId ~~OPTIONAL~~, protocolVersion ProtocolVersion, options Options, preferredMessageSize [5] IMPLICIT INTEGER, exceptionalRecordSize [6] IMPLICIT INTEGER, idAuthentication [7] ANY ~~OPTIONAL~~, -- see note below implementationId [110] IMPLICIT InternationalString OPTIONAL, implementationName [111] IMPLICIT InternationalString OPTIONAL, implementationVersion [112] IMPLICIT InternationalString OPTIONAL, userInformationField [11] EXTERNAL OPTIONAL, otherInfo OtherInformation OPTIONAL}	The **InitializeRequest** record is used to establish a session between a target and an origin. The parameters contained within **IntializeRequest** are: • The **referenceId** field is provided to enable tracking of a particular operation. A **referenceId** is assigned by the originator of the initialisation request operation and is returned in the initialisation response. • **protocolVersion** describes the version of the Z39.50 protocol supported. • **options** is a list of the set of services supported. • **preferredMessageSize** states the message size (in bytes) that the origin would prefer to use. • **exceptionalRecordSize** states the maximum message size that the client will accept for a present operation resulting in a single large record. If the **preferredMessageSize** and **exceptionalMessageSize** are the same, then the single record special case will not apply. • **idAuthentication** used by the origin and target to confirm the identity of the user (end user or retrieval manager acting on behalf of a user). The null value allows for a "guest" user with associated privilege. • **implementationId** used to identify the implementation used. • **implementationName** descriptive name of the implementation. • **implementationVersion** version of the implementation. • **userInformationField** is used to transfer user related information between an origin and a target. The definition of the user information is provided in UserInformation in the CIP Order ES (see Section 3.5.8.8). • **otherInfo** used to transfer other information between the target and origin (not used by the CIP).
ReferenceId ::= [2] IMPLICIT OCTET STRING	**ReferenceId** is a reference identifier assigned to an operation[6] (which is initiated by the origin of a request and terminated by the response of the target). It allows to track a particular operation by providing the means to match a request with its corresponding response. The format of the **ReferenceId** is specified in Appendix E.

[6] The CIP requires that all operations have a **`ReferenceId`**. The definition provided here is therefore valid for all the Z39.50 services described in the CIP.

ASN.1 Definition	Meaning

`IdAuthentication [7] CHOICE{`
`  open  VisibleString,`
`  idPass  SEQUENCE {`
`  groupId  [0]  IMPLICIT InternationalString OPTIONAL,`
`  userId  [1]  IMPLICIT InternationalString  OPTIONAL,`
`  password  [2]  IMPLICIT InternationalString OPTIONAL },`
`  anonymous  NULL,`
`  other  EXTERNAL`
`  -- use OID {Z39.50-AccessControl-prompt-1}`
	IdAuthentication contains parameters to identify the origin to the target. The parameters are:
	• **open** means that no access control is to be used.
	• **idPass** is a sequence consisting of
	  • **groupId**, the identifier for a group that the user is a member of. This is not applicable as the user is not envisaged to provide a group identifier.
	  • **userId**, the identifier for the user.
	  • **password**, the password of the user.
	• **anonymous** used to denote an anonymous user, the value of the parameter is null or empty. This allows an unregistered user to access a CIP retrieval manager with guest privileges.
	• **other** used by CIP clients to establish the identify of the CIP user. The format of this message is the same as that used for the access control service. This is to provide consistency with possible challenge/response sequences. Note that the target may still choose to challenge the identity of the origin under the CIP security approach for any subsequent operations requested by the target which require access rights other than “guest”.

ASN.1 Definition	Meaning

`InitializeResponse ::= SEQUENCE {`
`referenceId ReferenceId OPTIONAL,`
`protocolVersion ProtocolVersion,`
`options Options,`
`preferredMessageSize [5] IMPLICIT INTEGER,`
`exceptionalRecordSize [6] IMPLICIT INTEGER,`
`result [12] IMPLICIT BOOLEAN,`
`-- reject = FALSE; Accept = TRUE`
`implementationId [110] IMPLICIT InternationalString OPTIONAL,`
`implementationName [111] IMPLICIT InternationalString OPTIONAL,`
`implementationVersion [112] IMPLICIT InternationalString OPTIONAL,`
`userInformationField [11] EXTERNAL OPTIONAL,`
`otherInfo OtherInformation OPTIONAL}`
	InitializeResponse is generated by the target after reception of the **Initialize Request**. The response contains the following parameters;
	• The **referenceId** field is provided to enable tracking of a particular operation. A **referenceId** is assigned by the originator of the initialisation request operation and is returned in the initialisation response.
	• **protocolVersion** describes the version of the Z39.50 protocol supported.
	• **options** is a list of the set of services supported.
	• **preferredMessageSize** states the message size (in kilobytes) that the target will enforce.
	• **exceptionalRecordSize** states the maximum message size (in kilobytes) that the target will enforce for a present operation resulting in a single large record. If the **preferredMessageSize** and **exceptionalMessageSize** are the same, then the single record special case will not apply.
	• **result** indicates whether the target accepts the association or not, the values are “accept” and “reject”.
	• **implementationId** used to identify the implementation used.
	• **implementationName** descriptive name of the implementation.
	• **implementationVersion** version of the implementation.
	• **userInformationField** is used to transfer user related information between an origin and a target. For the definition of the user information, see **UserInformation** in Table 3-36.
	• **otherInfo** used to transfer other information between the target and origin.

`ProtocolVersion ::= [3] IMPLICIT BIT STRING{`
`version-1 (0), -- This bit should always be set, but does`
`-- not correspond to any Z39.50 version.`
`version-2 (1), -- "Version 2 supported."`
`-- This bit should always be set.`
`version-3 (2) -- "Version 3 supported."}`
	ProtocolVersion describes what version of the Z39.50 protocol is supported. For CIP the version of Z39.50 must be ‘version-3’. However, 'version-2' , with restricted functionality, is also supported by the CIP for interoperability with Z39.50 Version 2 based profiles.

ASN.1 Definition	Meaning

`Options ::= [4] IMPLICIT BIT STRING{`
`search (0),`
`present (1),`
`delSet (2),`
`resourceReport (3),`
`triggerResourceCtrl (4),`
`resourceCtrl (5),`
`accessCtrl (6),`
`scan (7),`
`sort (8),`
`-- (9) (reserved)`
`extendedServices (10),`
`level-1Segmentation (11),`
`level-2Segmentation (12),`
`concurrentOperations (13),`
`namedResultSets (14)}`
	Options describes what service options are requested (when the client or origin makes a request) or provided (when the server or target provides a response). The **Options** are encoded by setting bits in a bit string to identify which services are supported or requested. CIP-B targets and origins must respond to all requests (i.e. the bit for all services must be set) except the **scan**, **sort** and **level-1Segmentation** (since **level-2Segmentation** needs to be supported) services (for which the bit must not be set). Note that not all sites with CIP-B compliant targets must provide all of the services. Details on compliance levels of supplying services is provided in Appendix H.

3.5.2 Search Facility

The *Search service* is the means by which the *origin* (on behalf of the user) finds out what information is available.

This section describes how the Z39.50 *Search service* is used to fulfil the search requirements of the CIP.

The *Search service* permits the *origin* to perform the following two primary tasks:

- Search and identify product descriptors;
- Search and identify collection descriptors.

These in turn can either be ordered or used as *target* for subsequent searches.

A Z39.50 *Search* consists of a number of fundamental parts:

- The search query format (see Section 3.5.2.1);
- The *attributes* that can be used within the search query (see Section 3.5.2.2);
- The *target database* for the search (see Section 3.5.2.3);
- The *result set* created at the *target* that holds the results of the search (see Section 3.5.2.4).

Z39.50 is conventionally used to search flat database models where each record in a *target database* is homogeneous. In the CIP domain, however, the collection database has a hierarchical structure and the records are not homogeneous. Two types of searches, with different semantic meanings, are required: those to identify collections and those to identify product descriptors. Sections 3.9.2 and 3.9.4 include a description of how the Retrieval Manager must semantically process these two types of searches, and how the hierarchical nature of the database must be handled.

3.5.2.1 Search Queries

Z39.50 provides a choice of several query languages for the execution of search queries. The CIP shall support *Type-1* queries, which are general purpose set query structures defined using the Reverse-Polish-Notation (RPN). Note, however, that this does not preclude Retrieval Managers to support other additional Z39.50 query languages (i.e. all Retrieval Managers shall support **at least** *Type-1* queries, but they may optionally support other types of queries as well).

The *Type-1* query format is a very powerful syntactic notation which allows a great deal of flexibility in its use. *Type-1* queries themselves define very little semantic information about the query (only basic set operators are defined within the format), and so the semantics of a query must be defined by the application (this can be effectively done in the *Explain Database*, see Section 3.5.9.2.3).

The *Type-1* query use of RPN permits two operands to be logically[7] operated upon. In a *Type-1* query this is with the simple set operators **OR**, **AND** or **AND-NOT**[8]. Each of the operands are defined by an unordered and unstructured list of attributes. No restriction is imposed by the format on the content of

[7] The RPN *operators* are set operators (i.e. they result in the combination of sets). However, from a more abstract point of view, they are logical operators as they allow to formulate logical predicates (i.e. all the SAR images over the USA OR all the SAR images over Canada).

[8] The AND-NOT *Operator* is supported by the CIP. However, it should be used with great caution as queries including the AND-NOT *Operator* may be very inefficient and resource-consuming.

the attribute list, and an attribute[9] itself can be virtually anything defined by the application. In this way, the format allows the definition of queries that are syntactically correct, but which are semantically meaningless or absurd. It is therefore up to the application to define the permissible attributes in a query and which combinations of attributes are meaningful. The definition of the CIP *attribute set* (see Section 3.5.2.2 and Appendix A) and the definition of the CIP *attribute combinations* (see Appendix A.7) provide support for a CIP application to define the proper use of *Type-1* queries in the domain of the CIP.

So that the functionality of the basic Z39.50 *Type-1* query is clearly understood, a commented listing of the formal ASN.1 *Type-1* query definition from the Z39.50 specification is provided in Section 3.5.2.6. Note that, while the CIP supports the *Type-1* query format, not all the syntactical possibilities offered by the format are actually used by the CIP.

The following is an example of a *Type-1* query in the context of the CIP to illustrate its use.

Assume that a user wants to find all SAR images over Great Britain received after January 1, 1995. The user targets a search at a collection he knows of that contains all the images over Europe from a variety of instruments. He must therefore specify that the creation date must be later or equal to January 1 1995 and that the instrument that created the images must be a SAR. Moreover, he must specify the target spatial region so that only the images over Great Britain will be obtained. This is done, in this particular example, by defining a bounding rectangle that roughly contains Great Britain.

The query illustrated in Figure 3-1 would be created to perform this search. On the left is a tree structure showing the RPN structure of the query, on the right is the ASN.1 structure of the same query.

[9] Note that the definition of an *attribute* within Z39.50 is different, and much broader, than the CIP definition of an attribute (as described in the CIP Object Model in the ICS URD[URD]) where an attributes defines the semantics of an object. In Z39.50, an *attribute* can be anything that somehow specifies the semantics of a search query (i.e. an object, a relation between objects, the internal structure of an object, etc.).

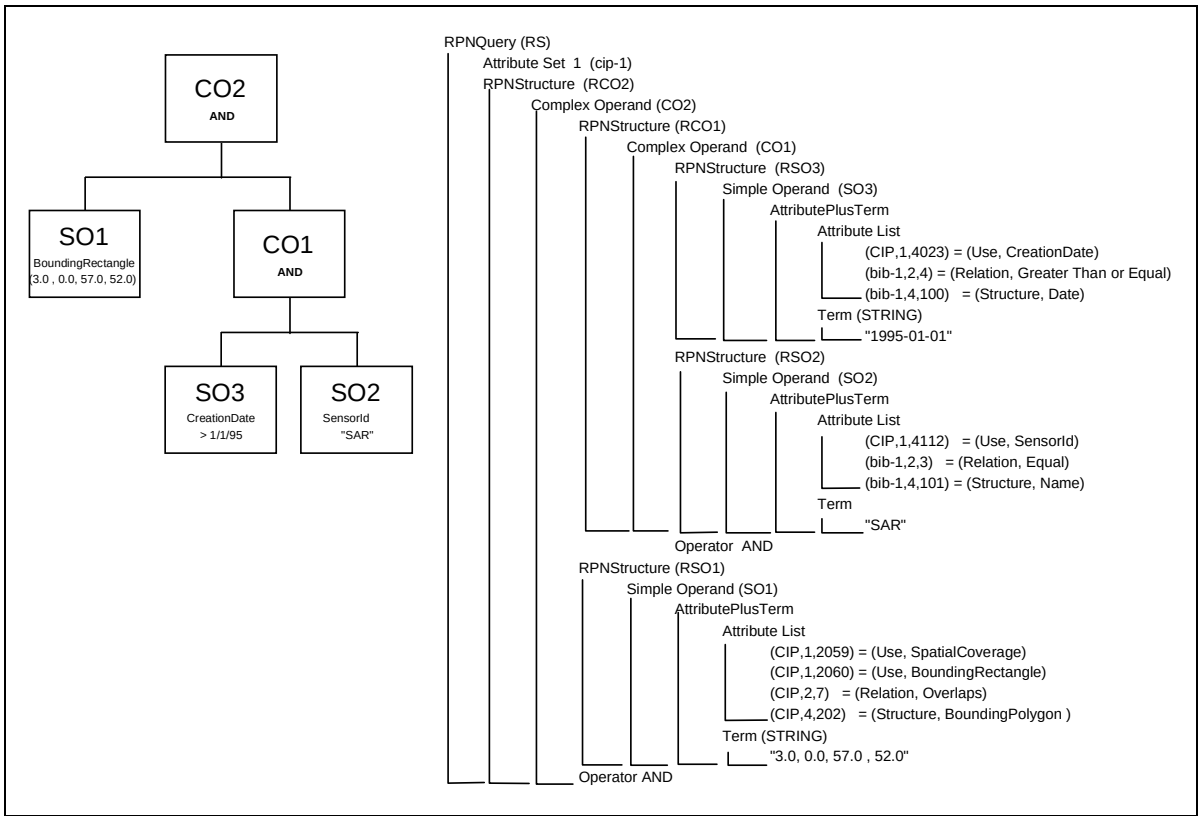

Figure 3-1: Example Z39.50 RPN Query

This query would be evaluated in the following manner:

1. The simple operand SO2 is evaluated. SO2 specifies a search attribute (*Use attribute* in Z39.50 terminology) of 'SensorId', the *term* supplied must be treated as a 'Name' (*Structure attribute* in Z39.50) and the value of the 'SensorId' *attribute* is checked to see if it is 'Equal' (*Relation attribute* in Z39.50) to the value of the *term*, which is 'SAR'. Those records in the *target database* that satisfy this simple operand (i.e. all SAR images over Europe) shall be named RSO2 (as in 'Results of Simple Operand 2') for our purposes.

2. Next, the simple operand SO3 is evaluated. SO3 specifies a *Use attribute* of 'CreationDate', the *term* supplied is treated as a 'Date' (*Structure attribute*) and the value of the 'CreationDate' *attribute* is checked to see if it is 'Greater Than or Equal' (*Relation attribute*) to the value of the *term*, which is '1995-01-01'. Those records in the *target database* that satisfy this simple operand (i.e. all images over Europe created after January 1 1995) shall be named RSO3 for our purposes (when a query such as this is actually processed, the intermediate result sets are not named and are not available to the user).

3. Next the complex operand CO1 is evaluated. The result of this operand is the records that appear both in *result set* RSO3 and *result set* RSO2, i.e. the intersection of the two sets is computed. This intermediate result we shall call RCO1. RCO1 is now the list of all SAR images over Europe after January 1 1995.

4. Next the simple operand SO1 is evaluated. SO1 specifies a compound *Use attribute* of 'SpatialCoverage, BoundingRectangle', the *term* supplied is treated as a 'BoundingPolygon' (*Structure attribute*) and the value of the 'SpatialCoverage, BoundingRectangle' is checked to see if it 'Overlaps' (*Relation attribute*) with the value of the *term*, which is externally defined and defines a rectangle with the South bounding coordinate of 52.0 North, the West bounding coordinate of 3.0 West, the North bounding coordinate of 57.0 North and the East bounding coordinate of 0.0 West. Those records in the *target database* that satisfy this simple operand (i.e. all images which overlap with the area defined in the query, which roughly covers Great Britain) shall be named RSO1 for our purposes.
5. Finally the complex operand CO2 is evaluated. The result of this operand is the records that appear both in *result set* RCO1 and *result set* RSO1, i.e. the intersection of the two sets is computed. The result of this complex operand, that we shall call RCO2, is therefore the list of all SAR images over Great Britain after January 1 1995.
6. The result of the overall RPNQuery, that we shall call RS, is identical to the *result set* RCO2 that has been obtained with the evaluation of the top level complex operand CO2.

3.5.2.2 Search Attributes

Z39.50 splits *attributes* into different types depending upon their function. The CIP uses the standard Z39.50 *attribute types* (defined for the bib-1 *attribute set*): *Use*, *Relation*, *Position*, *Structure*, *Truncation* and *Completeness*.

In a search query, the *Use attribute* identifies the *access point* against which the search *term* is to be matched. The *Structure attribute* identifies the form in which the search *term* has been supplied. The other types of *attributes* (*Relation*, *Position*, *Structure*, *Truncation* and *Completeness*) specify additional match criteria.

There is no restriction or formal guideline defining how the *attributes* must be used semantically in the syntax of the *attribute list* in a *Type-1* query. In particular, no ordering of the *attributes* or their types is imposed. Also, it is syntactically legal to have several *attributes* of the same *attribute type* (i.e. *Relation*, *Structure*) in an *attribute list* which would not be semantically meaningful (note however that it may be meaningful to have several *Use attributes* in the same *attribute list*). Therefore, rules for the definition of *attribute list* restrictions (which defines the permissible combinations of attributes) must be provided (see Appendix A.7 for the rules defining valid CIP *attribute combinations*).

The following table defines the *attribute types* supported by the CIP. The first column ('**#**') defines the value for each *attribute type*; this is required as in Z39.50 all *attribute types* are identified by a single number and not with a textual string. These values are only unique within the defined *attribute set*. For the CIP, the *Use attributes* are defined in the CIP *Attribute Set*, while all other types of *attributes* are imported from the bib-1 *attribute set*. In this manner, the OID of the CIP *Use attribute type* is defined as **`{Z39-50-CIP-Attribute-Set 1}`**, while the OIDs of the other *attribute types* are defined as **`{Z39-50-attributeSet 1 #}`**, and so, for instance, the OID of the CIP *Relation attribute* is defined as **`{Z39-50-attributeSet 1 2}`**. The last column indicates the appendix in which the full list of attributes of the indicated type can be found.

Note that, in addition to the standard CIP *attributes* defined above, each data provider may define its own *attribute set* for the definition of its local *attributes*. This is explained in more detail in Section 3.9.6.

Table 3-5: Z39.50 Attribute Types Summary

#	Type Name	Meaning	Appendix
1	**Use**	A ***Use*** *attribute* specifies an *access point*, i.e. an *attribute* that is searched, e.g. Sensor Name, Data Centre.	A.1
2	**Relation**	A ***Relation*** *attribute* describes the relationship of the *access point* (left side of the relation) to the search *term* (right side of the relation), e.g. equal, less than, inside.	A.2
3	**Position**	A ***Position*** *attribute* specifies the location that the search *term* must be looked for within the *field* or *sub-field* in which it appears, e.g. first in field, any position in field.	A.3
4	**Structure**	A ***Structure*** *attribute* specifies the data type of the search *term*, e.g. string, integer, date.	A.4
5	**Truncation**	A ***Truncation*** *attribute* specifies whether one or more characters may be omitted in matching the search *term* in the *target* system at the position specified by the *Truncation attribute*, right truncation, left truncation, no truncation.	A.5
6	**Completeness**	The ***Completeness*** *attribute* specifies that the contents of the search *term* represent a complete or incomplete field. *Completeness* indicates whether additional words must appear in a field or sub-field with the search *term*[10].	A.6

3.5.2.3 Database Names

There are a number of different *databases* with which a Retrieval Manager must interact. A Retrieval Manager is said to 'own' any of the *databases* that it manages and directly supports.

Every Retrieval Manager shall own, when appropriate, the following databases:

- An *Explain database*, which will provide the semantics of the information handled by the Retrieval Manager.
- An *Extended Services database*, which will provide access to the task packages created and managed by the use of the *Extended Services facility*.
- The CIP collections *database*, which will provide access to the database containing the definition of all the collections owned (and managed) by the Retrieval Manager.
- The collection *databases*, which correspond to all the collections within the Retrieval Manager's domain (i.e. all the collections which are directly managed by the Retrieval Manager and therefore contained in the CIP collections *database*)[11].
- The user *database*, which contains the user profiles managed within the Retrieval Manager's domain.[12]

The format of the various *database* names is formally specified in Appendix E.

---

[10] Note the difference between the *Truncation* and *Completeness attribute types*: a *Truncation attribute* specifies whether characters may be omitted in matching a search *term*. On the other hand, a *Completeness attribute* specifies the level at which the search *term* must be matched. In other words, both *attribute types* allow partial matching, but this is achieved in two different ways. A *Truncation attribute* allows to perform partial matching by truncating an *element* (i.e. 'search the *database* for all the records which contain a word starting with 'pre''), whereas a *Completeness attribute* allows to define the granularity at which a matching must be performed (i.e. 'search the *database* for all the records which contains a phrase containing the word 'preamble'').

[11] At present collection *databases* are the only type of *database* holding CIP data. It can be envisaged that other types of *databases* may be added in the future.

[12] Note that not all Retrieval Managers are required to own a user *database*.

3.5.2.4 Result Set Names

Whenever the *origin* initiates a search, a *Result-set-name* must be defined so that the *result set* can be reused later in a *Present* or *Search service*. The *Result-set-name* is defined by the *origin* and shall follow the format defined in Appendix E.

Note that all *result sets* are deleted by the *target* at the end of the *Z-association*. The result sets may also be explicitly deleted at any time during a Z-Association using the Result-set delete Facility.

3.5.2.5 CIP Search Targeting and Scope

Any Z39.50 *search request* must identify a *target database* for a search. Although not specifically specified, Z39.50 *databases* can be assumed to be flat. For the CIP, on the other hand, collections are defined hierarchically. Moreover, there are two different types of searches which a CIP user may wish to perform:

- Find collections of interest for further searching (i.e. a collection search).
- Find individual product descriptors which may eventually lead to the order of an actual product (i.e. a product search).

Therefore as the CIP data model has a more complex structure than the data model used by Z39.50 and different types of searches are identified, enhancements to the Z39.50 search targeting are required for the CIP. This is achieved using the following *database* definitions (see also Section 3.5.2.3):

- Each Retrieval Manager contains a collections *database*, which contains the definition of all the collections that the Retrieval Manager owns and can be targeted for a search.
- Specific collections in the CIP Collection hierarchy are equivalent to logical *databases*, that is each collection is seen as a single logical *database* which can be targeted for a search.

The two types of searches are realised in the following manner:

- To perform a collection search, the CIP Collections database is targeted. The selection of the specific collection within the collection hierarchy at which the search is targeted is specified within the RPN query, as described in Section 3.9.2.
- To perform a product search, a specific collection within the product collection hierarchy is targeted (see also Section 3.9.3).

The *origin* may also explicitly request the various types of searches using the **`otherInfo`** field within the search object[13]. This is presented in more detail in Appendix E.

As CIP searches may be forwarded to remote Retrieval Managers (because a collection can point to remote collections at other Retrieval Managers), the length of time to complete a search may be significant for a user. There is therefore the need in the CIP to be able to limit the scope of the search to the collection or product descriptors that are owned by the local Retrieval Manager only, or alternatively, to specify that the search may be passed onto all members collections, irrespective of the owning Retrieval Manager.

The definition of the scope of a search in the CIP is also achieved using the **`otherInfo`** field within the search object, which allows to define whether a search shall be executed locally (i.e. the default behaviour), or whether it may be executed at a remote Retrieval Manager. This is presented in more detail in Appendix E.

[13] Note that whilst the use of **`otherInfo`** is optional and not necessary for the determination of the type of search by the target, it is recommended that CIP *origins* do use it.

This principle of the choice of the type and scope of a search is demonstrated with the use of Figure 3-2 below:

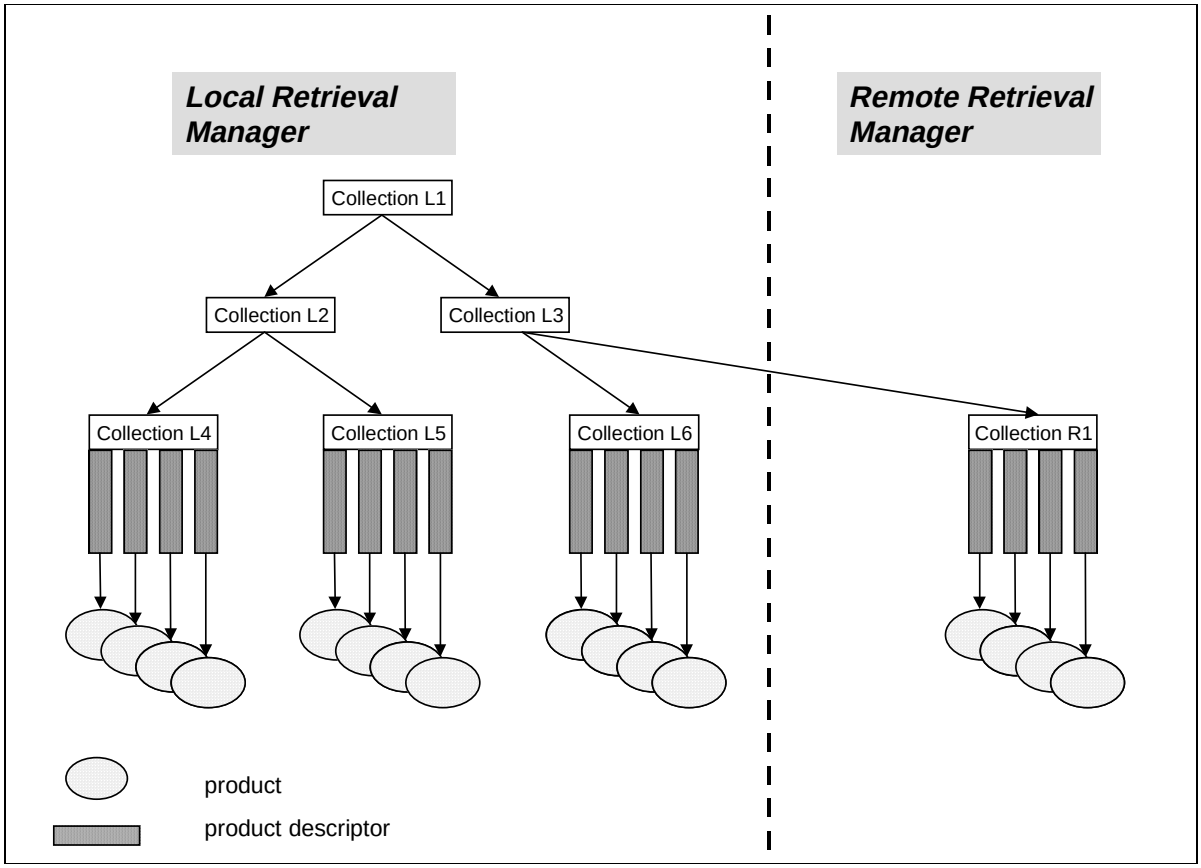

Figure 3-2: Search Types and Scopes Example

If a collection search was directed at collection L1, then the attributes (see Section 3.5.2.2) associated with all collections below this node (collections L1,L2, L3, L4, L5, L6, , and R1) shall be searched. The exact content of the response will be as defined by the user, but the information shall include the collection node identifiers plus the collection attributes as selected by the user, for those collections that fulfil the search requirements. Note, no searching or other reference to product descriptor attributes will be made. If the same search is direct at collection node L3, then only collection nodes L3, L6 and R1 will be searched.

If a product descriptor search was indicated and it was directed at collection node L1, then the collection tree structure itself is immaterial, the only significance being which product descriptors are found within the terminal collections at the leaves below the target, in this case collections L4, L5, L6 and R1. The search is directed to these terminal collections and the attributes of the product descriptors listed as members of the collections are searched. Note that the collections attributes themselves are not searched.

The following table summarises the results from targeting a search at collection L1 with the various search control options:

	local search	**wide search**
collection search	Collections attributes from: **L1, L2, L3, L4, L5, L6**	Collection attributes from: **L1, L2, L3, L4, L5, L6, R1**
product search	Attributes of product descriptors in collections:	Attributes of product descriptors in collections:

	L4, L5, L6	**L4, L5, L6, R1**

3.5.2.6 Search Service Formal Definitions

The *Search facility* consists of a single service, the *Search service*, presented in the subsections below.

3.5.2.6.1 Search Request

The use of the *search request* by the CIP is presented in detail in Table 3-6.

Table 3-6: Search Request Usage

ASN.1 Definition	Meaning

`SearchRequest ::= SEQUENCE{`
`referenceId ReferenceId ~~OPTIONAL~~,`
`smallSetUpperBound [13] IMPLICIT INTEGER,`
`largeSetLowerBound [14] IMPLICIT INTEGER,`
`mediumSetPresentNumber [15] IMPLICIT INTEGER,`
`replaceIndicator [16] IMPLICIT BOOLEAN,`
`resultSetName [17] IMPLICIT InternationalString,`
`databaseNames [18] IMPLICIT SEQUENCE OF DatabaseName,`
`smallSetElementSetNames [100] ElementSetNames OPTIONAL,`
`mediumSetElementSetNames [101] ElementSetNames OPTIONAL,`
`preferredRecordSyntax [104] IMPLICIT OBJECT IDENTIFIER OPTIONAL,`
`query [21] Query,`
`additionalSearchInfo [203] IMPLICIT OtherInformation OPTIONAL,`
`otherInfo OtherInformation OPTIONAL}`
	The **SearchRequest** object defines the elements transferred for any search operation and contains the following elements:
	• **referenceId** is a reference identifier assigned to the operation (consisting of a SearchRequest and a SearchResponse) initiated by the SearchRequest which allows to match a request with a response.
	• A **smallSetUpperBound** is the small result set limit, determined by **resultCount**. When collections are searched and a **SearchRequest** is therefore forwarded down the collection hierarchy, the piggybacking parameters **smallSetUpperBound**, **largeSetLowerBound** and **mediumSetPresentNumber** are simply forwarded, without any modification.
	• A **largeSetLowerBound** is the large result set limit, determined by **resultCount**.
	• A **mediumSetPresentNumber** is the maximum number of response records required when **resultCount** falls between the small and large set bounds.
	• A **replaceIndicator** flag is used to indicate the action to take on supplying a **resultSetName** which already exists. If the flag is 'off,' the existing result set is unchanged and an error diagnostic occurs. If the flag is 'on', the existing set is deleted and a new result set with the given name is used.
	• A **resultSetName** specifies a name for the result set (to be created by the target), for subsequent reference within the Z-association.
	• **databaseNames** is used to indicate by the origin the databases (to which the search query applies. In the CIP domain, the search query will be target at collections or the Explain database (see also Section 3.9).
	• **smallSetElementSetNames** are used to specify the names of small element sets defining the preferred composition of records within a small result set.
	• **mediumSetElementSetNames** are used to specify the names of large element sets defining the preferred composition of records within a medium result set.
	• A **preferredRecordSyntax** can be specified for the preferred record syntax for retrieval

ASN.1 Definition	Meaning
	records. • The **query** is the search query type defined by **Query**. For the CIP Release B the Type-1 query syntax must be supported. • **additionalSearchInfo**, which may be used to provide information related to the search process. • **otherInfo** may be used by the CIP origin to specify the scope of a search, i.e. whether the search domain is wide or restricted to a local search. This is achieved using the CIPSpecificInfo EXTERNAL and selecting searchControl, as defined in Appendix E.6.1. If **otherInfo** is not provided, the type of item descriptors to be searched shall be derived from the query definition and/or the content of the collection and the default scope of a local search shall be assumed. (See Section 3.5.2.5 for more details). Although not mandatory, it is recommended that **otherInfo** be provided.
`DatabaseName ::= [105] IMPLICIT InternationalString`	**DatabaseName** is the identifier of a database, i.e. a collection descriptor. The format of a **DatabaseName** is specified in Appendix E.

`Query ::= CHOICE{`
`type-0 [0] ANY,`
`type-1 [1] IMPLICIT RPNQuery,`
`type-2 [2] OCTET STRING,`
`type-100 [100] OCTET STRING,`
`type-101 [101] IMPLICIT RPNQuery,`
`type-102 [102] OCTET STRING}`
	For the CIP Release B **Type-1** queries are the only mandatory **Query** to be supported.
	In Z39.50 Version 3, **Type-1** queries are extended RPN queries, an extension to the Version 2 **Type-1 Query** to allow proximity searching and restriction of result sets by attributes.
	• **Type-0** to be used only on prior arrangement between client and server.
	• **Type-2** is the ISO8777 (ISO CCL) type query (not recommended for reasons of grammar extensibility restrictions).
	• **Type-100** is the Z39.58 type query. (Closely related to **Type-2** queries, with the same problems)
	• **Type-101** is identical to **Type-1** queries for Z39.50 Version 3. **Type-1** queries for Version 2 do not include proximity and result set restriction.
	• **Type-102** still being defined, e.g. to incorporate Ranked List queries.

`RPNQuery ::= SEQUENCE{`
`attributeSet AttributeSetId,`
`rpn RPNStructure}`
	An **RPNQuery** consists of an **AttributeSetId**, which identifies the default attribute set used in the query (this may be overridden on a per attribute basis), and an **RPNStructure**, which is a tree structure depicting the query. Nominally **AttributeSetId** must be the CIP Use attribute set identified by OID {Z39.50-CIP-AttributeSet} and described in Appendix A. However, CIP targets shall not reject other **AttributeSetId**s and must attempt to satisfy the query solely on the attribute value in the **AttributeElement** structure.

ASN.1 Definition	Meaning

`RPNStructure ::= CHOICE{`
`  op  [0] Operand,`
`  rpnRpnOp  [1] IMPLICIT SEQUENCE{`
`  rpn1  RPNStructure,`
`  rpn2  RPNStructure,`
`  op  Operator }}`
	The **RPNStructure** consists of an **Operand**, the **Operand** represents a set of database records.
	There are two kinds of **Operands**:
	• Simple **Operands** are the leaf nodes (i.e. **Operands** which cannot be further decomposed) of a query branch and represent an actual set of database records. This may be the name of an existing result set or a list of attributes to be evaluated and hence resulting in a set of database records.
	• Complex **Operands** are the non-leaf nodes of a query and must be further evaluated to result in a simple **Operand**. That is, it consists of two further **Operands** combined through a Set operation defined by an **Operator**.
	A Complex **Operand** is a branch of the query tree whose root is an **Operator**, and which contains two branches: a left **Operand** and a right **Operand**. The two sub-trees of a complex **Operand** are **RPNStructures** themselves, therefore resulting in a recursive definition of the tree structure.

`Operand ::= CHOICE{`
`  attrTerm  AttributesPlusTerm,`
`  resultSet  ResultSetId,`
`  resultAttr  ResultSetPlusAttributes}`
	A simple **Operand** directly represents a set of database records, and is one of the following:
	• An **AttributePlusTerm** that represents the set of database records obtained by the evaluation of an **AttributeList** and a **Term** against the database(s) specified in the overall Search Request.
	• A **ResultSetId** that identifies a set of database records that was created as the result of a previous search query (in other words a persistent result set).
	• A **ResultSetPlusAttributes** that represents the set of database records represented by evaluating an existing result set against an attribute list. This requires the use of the extended result set model.

`AttributesPlusTerm ::= [102] IMPLICIT SEQUENCE{`
`  attributes  AttributeList,`
`  term  Term}`
	An **AttributePlusTerm** consists of an **AttributeList** and a **Term**. This forms the heart of a single query on a item descriptor attribute, e.g. to see if attribute x is less than value (term) y.

`ResultSetId  ::=  [31] IMPLICIT InternationalString`	A **ResultSetId** is the identifier of a result set. The format of a **ResultSetId** is specified in Appendix E.

`ResultSetPlusAttributes ::= [214] IMPLICIT SEQUENCE{`
`  resultSet  ResultSetId,`
`  attributes  AttributeList}`
	A **ResultSetPlusAttribute** consists of a result set, identified by a **ResultSetId**, and an **AttributeList** which restricts the result set in some manner.
	This **ResultSetPlusAttribute** represents one of the following[14]:
	• A set of database records directly identified by the **ResultSetId** and restricted by the **AttributeList**. In other words, if there was a set of result records S1, from database D1 and it was restricted by the an **AttributeList** to yield another set of result records S2 (obviously also from the database D1), then S2 would be a subset of S1.
	• A set of database records indirectly identified by the **ResultSetId** and restricted by the

[14] See Appendix 13 'ERS: Extended Result Set Model' of the Z39.50 specification[Z3950] for exact details of this construct.

ASN.1 Definition	Meaning
	AttributeList. In other words, if there was a set of result records S1 from database D1, the **AttributeList** would be applied against this result set, to identify a set of result records S2 from database D2 by virtue that the two databases have a common attribute from the **AttributeList**.
`AttributeList ::= [44] IMPLICIT SEQUENCE OF AttributeElement`	An **AttributeList** is a list of **AttributeElements**. The **AttributeList** must contain a permissible CIP attribute combination (see Appendix A.7).

`Term ::= CHOICE{`
`general [45] IMPLICIT OCTET STRING,`
`numeric [215] IMPLICIT INTEGER,`
`characterString [216] IMPLICIT InternationalString,`
`oid [217] IMPLICIT OBJECT IDENTIFIER,`
`dateTime [218] IMPLICIT GeneralizedTime,`
`external [219] IMPLICIT EXTERNAL,`
`integerAndUnit [220] IMPLICIT IntUnit,`
`null [221] IMPLICIT NULL}`
	A **Term** is the instance value that is used for comparison against the target database records, it can be of one of the following types:
	• A **general Term**, which represents a string of 8-bit bytes.
	• A **numeric Term**, which represents an integer value.
	• A **characterString Term**, which represents a string of ASCII characters.
	• An **oid**, which refers to an ISO object defined elsewhere.
	• A **dateTime Term**, which represents a date and/or time as defined by the ASN.1 base type GeneralizedTime.
	• An **external Term**, which refers to an ASN.1 type defined externally to Z39.50 (i.e. a profile specific type).
	• An **integerAndUnit Term**, which represents an integer value together with its unit.
	• The **null Term**, which represents an empty element (this can be used when NO **Term** is to be used, i.e. with the **Any** use attribute).

`Operator ::= [46] CHOICE{`
`and [0] IMPLICIT NULL,`
`or [1] IMPLICIT NULL,`
`and-not [2] IMPLICIT NULL,`
`prox [3] IMPLICIT ProximityOperator}`
	The **Operator** determines how the two sets of records that are obtained as a result of evaluating both **Operands** of a **Complex Operand** are combined into a resulting set of records.
	The set of database records (S) represented by the result set of the complex **Operand** (R) is equivalent to: the set of database records (S1) represented by the result set of the left **Operand** (R1) and the set of database records (S2) represented by the result set of the right **Operand** (R2) combined by the **Operator**, which may be one of the following:
	• **AND** S is the intersection of S1 and S2.
	• **OR** S is the union of S1 and S2.
	• **AND-NOT**[15] S is the set of elements in S1 which are **not** in S2.
	• **PROX** S is as defined in the explanation below.

[15] The AND-NOT *Operator* is supported by the CIP. However, it should be used with great caution as queries including the AND-NOT *Operator* may be very inefficient and resource-consuming.

ASN.1 Definition	Meaning

`AttributeElement ::= SEQUENCE{`
`  attributeSet [1] IMPLICIT AttributeSetId OPTIONAL,`
`  attributeType [120] IMPLICIT INTEGER,`
`  attributeValue CHOICE{`
`  numeric [121] IMPLICIT INTEGER,`
`  complex [224] IMPLICIT SEQUENCE{`
`  list [1] IMPLICIT SEQUENCE OF StringOrNumeric,`
`  semanticAction [2] IMPLICIT SEQUENCE OF INTEGER OPTIONAL}}}`
	An **AttributeElement** uniquely identifies an attribute. It consists of an optional **AttributeSetId**, which identifies the attribute set where the attribute is defined (which, if used, overrides the default **AttributeSetId** defined in the highest level **RPNQuery** structure), an **attributeType**, which identifies the type of the attribute (i.e. Use, Structure, Relationship, etc., see Section 3.5.2.2) and an **attributeValue**, which identifies the particular attribute in the attribute set (note, this is not the instance value of the attribute, but the value of the number assigned to the attribute in the attribute set definition).
	
	An **AttributeElement** therefore is uniquely identified by the attribute set in which it belongs (either implicitly by using the default attribute set or explicitly), its type within the attribute set, and its numeric value within the attribute type.
	
	An **attributeValue** can be either simple or complex:
	• A simple **attributeValue** is a numeric value which uniquely identifies the attribute.
	• A complex **attributeValue** consists of a list of attributes, each identified by a value, and an optional list of **semanticActions**. The list of attribute values identifies a set of attributes which have a similar meaning (i.e. can be treated as synonyms), and the **semanticAction** defines how the target must handle the synonymous attribute values (i.e. specifies how the target must select one of the values for the evaluation of a query).

`ProximityOperator ::= SEQUENCE{`
`  exclusion [1] IMPLICIT BOOLEAN OPTIONAL,`
`  distance [2] IMPLICIT INTEGER,`
`  ordered [3] IMPLICIT BOOLEAN,`
`  relationType [4] IMPLICIT INTEGER{`
`  lessThan (1),`
`  lessThanOrEqual (2),`
`  equal (3),`
`  greaterThanOrEqual (4),`
`  greaterThan (5),`
`  notEqual (6)},`
`  proximityUnitCode [5] CHOICE{`
`  known [1] IMPLICIT KnownProximityUnit,`
`  private [2] IMPLICIT INTEGER}}`
	The evaluation of a complex **Operand** with the **ProximityOperator** depends on the type of each **Operand**.
	
	If both **Operands** are **AttributePlusTerm Operands**, S is the subset of records in the set (S1 AND S2) for which (left **Operand**-**ProximityOperator**-right **Operand**) is true.
	
	Otherwise, result records R1 and R2 must conform to the extended result set model for proximity, and thus contain position information in the form of position vectors. For each record represented by both R1 and R2, every ordered pair consisting of a position vector associated with the record as represented in R1 and a position record as represented in R2 is considered. For each pair that qualifies according to the proximity test, the record is qualified into the set S and a position vector is created for that record as represented in the resultant set and composed from that ordered pair.
	
	The **ProximityOperator** is composed of the following elements:
	• The **exclusion** flag, which is optional, is used to negate the **ProximityOperator**.
	• The **distance** is the difference between the ordinal values of the two **Operands** tested with the **ProximityOperator**.
	• The **ordering** flag indicates the direction of the proximity test. If the flag is set, the proximity test is for 'right' proximity only (the ordinal value of the left **Operand** must not exceed the ordinal value of the right **Operand** and **distance** is compared with the difference between the right and left ordinal values). Otherwise, the proximity test is for 'right' or 'left' proximity. In other words, **distance** is compared with the absolute value of the difference between the left and right ordinal values.

ASN.1 Definition	Meaning
	• The **relationType** is the type of relation between the left and right ordinal values and is one of 'less than', 'less than or equal', 'equal', 'greater than or equal' or 'not equal'. • The **proximityUnitCode** is the unit which is used for the proximity test. The **proximityUnitCode** is either a **KnownProximityUnit** or a privately defined unit.
KnownProximityUnit ::= INTEGER{ character (1), word (2), sentence (3), paragraph (4), section (5), chapter (6), document (7), element (8), subelement (9), elementType (10), byte (11)}	The **KnownProximityUnits** used in the proximity test are: **Character, Word, Sentence, Paragraph, Section, Chapter, Document, Element, Sub-element, Element Type, Byte**

3.5.2.6.2 Search Response

The use of the *search response* by the CIP is presented in detail in Table 3-7.

Table 3-7: Search Response Usage

ASN.1 Definition	Meaning

SearchResponse ::= SEQUENCE{
referenceId ReferenceId ~~OPTIONAL~~,
resultCount [23] IMPLICIT INTEGER,
numberOfRecordsReturned [24] IMPLICIT INTEGER,
nextResultSetPosition [25] IMPLICIT INTEGER,
searchStatus [22] IMPLICIT BOOLEAN,
resultSetStatus [26] IMPLICIT INTEGER{
subset (1),
interim (2),
none (3)} OPTIONAL,
presentStatus PresentStatus OPTIONAL,
records Records OPTIONAL,
-- Following two parameters may be used only if version 3 is
-- in force.
AdditionalSearchInfo [203] IMPLICIT OtherInformation
~~OPTIONAL~~,
otherInfo OtherInformation OPTIONAL}
	The **SearchResponse** contains the response to a **SearchRequest**. It contains the following information:
	• **referenceId**, which is the reference identifier of the operation initiated by the **SearchRequest**.
	• **resultCount**, which is the number of database records identified by the result set. When collections are searched, the result set has an internal nested structure which reflects the hierarchical structure of the searched collection. **resultCount** is the total number of records at the leaf level of the result set. At each node, the target must therefore consolidate the **resultCount** forwarded back from **SearchResponse** to the **SearchRequest** that were originally forwarded from the node.
	• **numberOfRecordsReturned**, which is the total number of records returned in the **SearchResponse**. Note that if piggybacking is not requested, zero records are returned. When collections are searched, **numberOfRecordsReturned** is the total number of records at the leaf level of the result set that are returned in the **SearchResponse**. At each node, the target must therefore interpret which **records** must be forwarded back in response to the originating **SearchRequest** by interpreting the piggybacking parameters which were included in the **SearchRequest**.
	• **nextResultSetPosition**, which is the position of the next result set item to be retrieved.

ASN.1 Definition	Meaning
	• **searchStatus**, which is the status of the search and is either "success" (i.e. the search completed successfully) or "failure" (i.e. the search did not complete successfully). Note that "success" does not imply that the expected records are being returned as part of the response, nor that any database records met the search criteria (i.e. a successful search may return an empty result set). "failure", on the other side, does imply that none of the expected response records is being returned. It also implies that one or more diagnostic record(s) is provided to explain the failure. • **resultSetStatus**, which is the status of the result set when the **searchStatus** is 'failure'. The value of the resultSetStatus is either **subset** (if partial valid results are available), **interim** (if partial results, not necessarily valid, are available), and **none** (if no result set is available). • **presentStatus**, indicates the status regarding the retrieval of records. It is included only is searchStatus is "success". For more detail, see Table 3-12. • **records**, which are the records returned by the SearchResponse (i.e. the piggybacked records). For more detail, see Table 3-12 • **additionalSearchInfo**, is used to provide information related to the search process. CIP B uses {Z39.50-UserFormat-searchResult-1}, which is described in detail in Section 3.5.6.4. • **otherInfo**, may be used to return the type of the search if provided in the **SearchRequest**.

3.5.3 Retrieval Facility

This section describes the components and procedures used by CIP to return records that have been located through execution of the *Search service* as defined in Section 3.5.2.

3.5.3.1 Introduction

Once a CIP *search request* has been submitted to the Retrieval Manager, performed at the *target(s)*, and has responded successfully, a *result set* is made available for subsequent *retrieval requests* by the *origin*[16].An *origin* may request *piggybacking* in a CIP *search request*, in which case a small number of *retrieval records* are returned in the *search response* itself. However, the most common way to retrieve records is to use the *Retrieval Facility*.

The *Retrieval Facility* allows the client to request *retrieval records* that correspond to *database records* represented by a named *result set*. *Database records* are referenced by their relative position within the *result set*. The Retrieval Facility consists of the following services:

- *Present service*, which contains the *Present request* and *Present response*
- *Segment service*, which contains the *Segment request*.

Using the *Present request*, the *origin* can specify a range of records to be retrieved from the *result set* (by specifying a starting position in the *result set* and the number of records to be retrieved from that position). This may follow through with subsequent *Present requests* specifying different ranges. The CIP gives the client great flexibility in how it requests that the *result set* be formatted; the client may choose basic default formats or complete customisation.

The *target* responds to the *Present request* with a *present response*, which contains the requested *retrieval records*. Two kinds of *present responses* are distinguished, depending on whether *segmentation* is in effect or not:

- *Simple present response*

  A *simple present response* occurs when no *segmentation* is in effect, i.e. the requested *retrieval records* fit into a single *segment*. This single *segment* is contained in a single *Present response*.

- *Aggregate present response*

  An *aggregate present response* occurs when *segmentation* is in effect, i.e. the requested *retrieval records* do not fit into a single *segment*. The *target* thus segments the response by sending one or more *Segment requests* before concluding the response with a *Present response*.

Each *Segment request* and *Present response* is referred to as a *segment* of a *present response*.

To illustrate this, consider a search for products. An *origin* targets its search to a particular search domain by targeting a specific collection. This defines a set of product descriptors that can potentially match the search. By defining search criteria, as described in Section 3.5.2.1, and applying them to the search domain, the client obtains a subset of the original domain as the result of its query. This result is known as *result set* and contains a set of *result records*. It is those *result records* that may be retrieved, in the form of *retrieval records*, by the *Retrieval facility*. Using the *Present request*, the *origin* specifies the subset of the *result set* that is requested to be retrieved. The *target* then responds with a *present response*. The *present response* consists either of a single *segment* containing all the requested *retrieval records* to be returned by the *simple present response* or a number of *segments*,

[16] This specification does not define specific client concepts or operations, but it can be assumed that any client software will request query results on behalf of the user of that client.

each containing a subset of the *retrieval records*, to be returned by the *aggregate present response*. This process is illustrated in Figure 3-3:

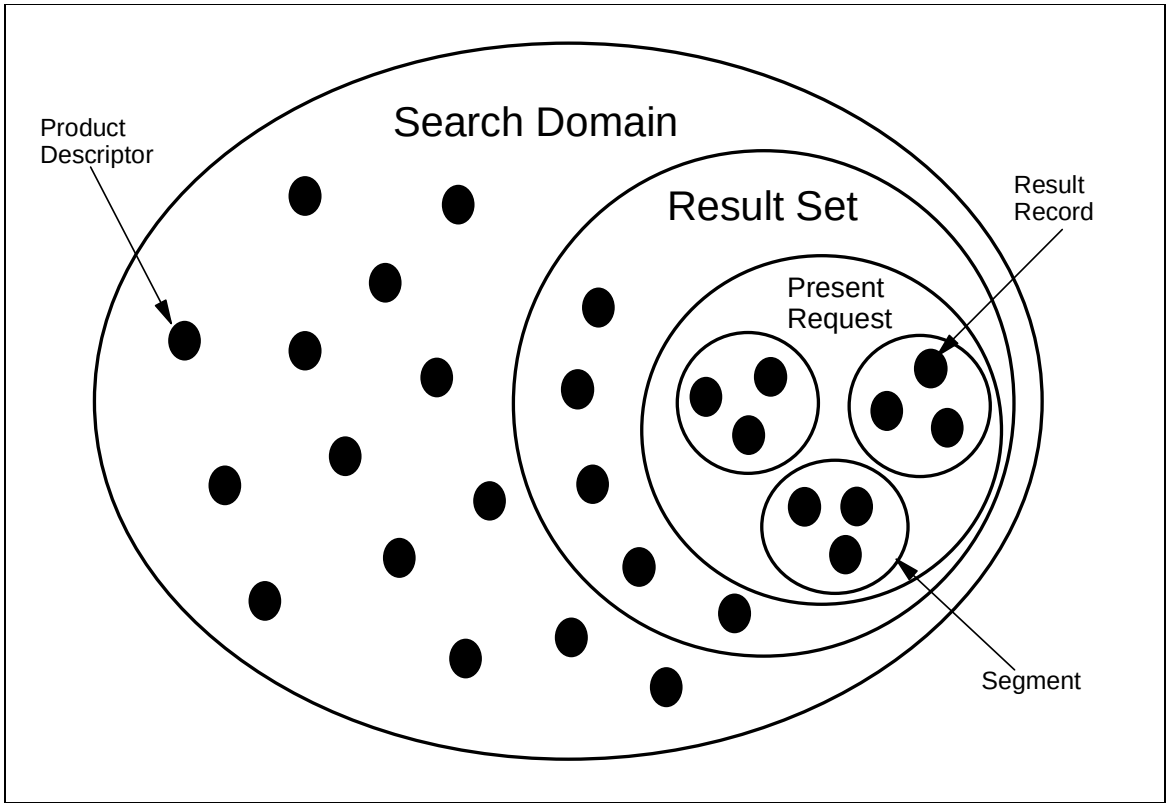

Figure 3-3: Search and Results Retrieval

3.5.3.2 Database Schema

This section provides a description of the Z39.50 *database schemas* concepts and how those concepts are customised for CIP purposes. It is divided in the following two subsections:

- Section 3.5.3.2.1 provides a general introduction of the Z39.50 concepts associated with the *Retrieval facility*, in particular the definition of *database schemas* in order to provide a framework for the understanding of the definition of the CIP *database schemas*. However, for a more detailed and comprehensive description, the Z39.50 Specification[Z3950], Appendix 12:'TAG' and Appendix 14:'RET' should be consulted.
- Section 3.5.3.2.2 provides a description of the CIP *database schemas* and introduces their use.

3.5.3.2.1 Z39.50 Database Schemas Concepts

Associated with a *database* are one or more *database schemas*, which provide a common understanding between the client and server on the information contained in the records of the *database*.

The definition of a *schema* is performed according to the following principles:

- A *schema* consists of *schema elements*, each of which corresponds to an individual field of a *target* database record;
- A *schema element* is defined in terms of *elements*;
- An *element* is identified by a *tag* in a *tag set*;

- A *tag* consists of a *tag type* and a *tag value*. The *tag type* qualifies the *tag value* and, within the CIP, identifies the *tag set* to which the *tag* belongs. The *tag value* is therefore the unique identifier of the *element* within its *tag type* (see example in Table 3-8).
- An *abstract record structure* is the primary component of a *schema*. The *abstract record structure* performs the mapping between the *elements* from the *tag sets* and the *schema elements* in the following manner:
  - The *abstract record structure* lists all the *schema elements* in a *schema* and supplies information associated with each *schema element*, specifying its definition, whether it is mandatory, and whether it is repeatable.
  - Each *schema element* in an *abstract record structure* is defined using a *tag path*. A *tag path* is a representation of the hierarchical path of a *schema element* within a *schema*, expressed as a sequence of nodes in a tree, each represented by a *tag* (corresponding to an *element* in a *tag set)* (see example in Table 3-9).

To illustrate this, consider the following (simplified) example:

A *tag set* for 'Coordinates' is defined. It is identified by the *tag type* 4. This *tag set* contains three *tag values*: Point has value 4071, Latitude has value 3118 and Longitude has value3119. Thus, the following three *elements* are defined, each with its unique *tag*:

Table 3-8: Tag Set Definition

Tag		Element Name
Tag Type	Tag Value	
4	4072	Point
4	3118	Latitude
4	3119	Longitude

From these *elements*, a *schema* can be defined using the following *abstract record structure*:

Table 3-9: Abstract Record Structure Definition

Abstract Record Structure				
Schema Element Tag Path	Schema Element Name	Recommended Data Type	Mandatory?	Repeatable?
(4 , 4072)	Point	(structured)	Yes	Yes
(4 , 4072) / (4 , 3118)	PointLatitude	InternationalString	No	No
(4 , 4072) / (4 , 3119)	PointLongitude	InternationalString	No	No

3.5.3.2.2 CIP Database Schemas

This section describes the various CIP specific definitions necessary for the CIP *database schemas*.

The *elements* necessary for building the CIP schemas are defined in a unique *tag set*, the CIP *tag set*. This *tag set* is defined in detail in Appendix B.

The following three *database schemas* are defined for the CIP:

- The **Collection Schema** contains the *schema elements* required to describe a collection.
- The **Product Schema** contains the *schema elements* required to describe a product.
- The **User Schema** contains the *schema elements* required to describe a user.

The CIP *schemas* are defined in detail using an *abstract record structure* in Appendix C.3.

Note that some *elements* defined in the *tag sets* (and used as *schema elements* in the *database schemas*) are also defined as *Use attributes* (see Section 3.5.2.2 and Appendix A.1). Those *elements*

are the searchable *database elements*, i.e. in the CIP profile the *Use attributes* are a subset[17] of the *elements* defined in the *tag sets*. This relationship between the *elements* defined in the CIP *tag set* and the CIP *Use Attributes* is illustrated in Figure 3-4:

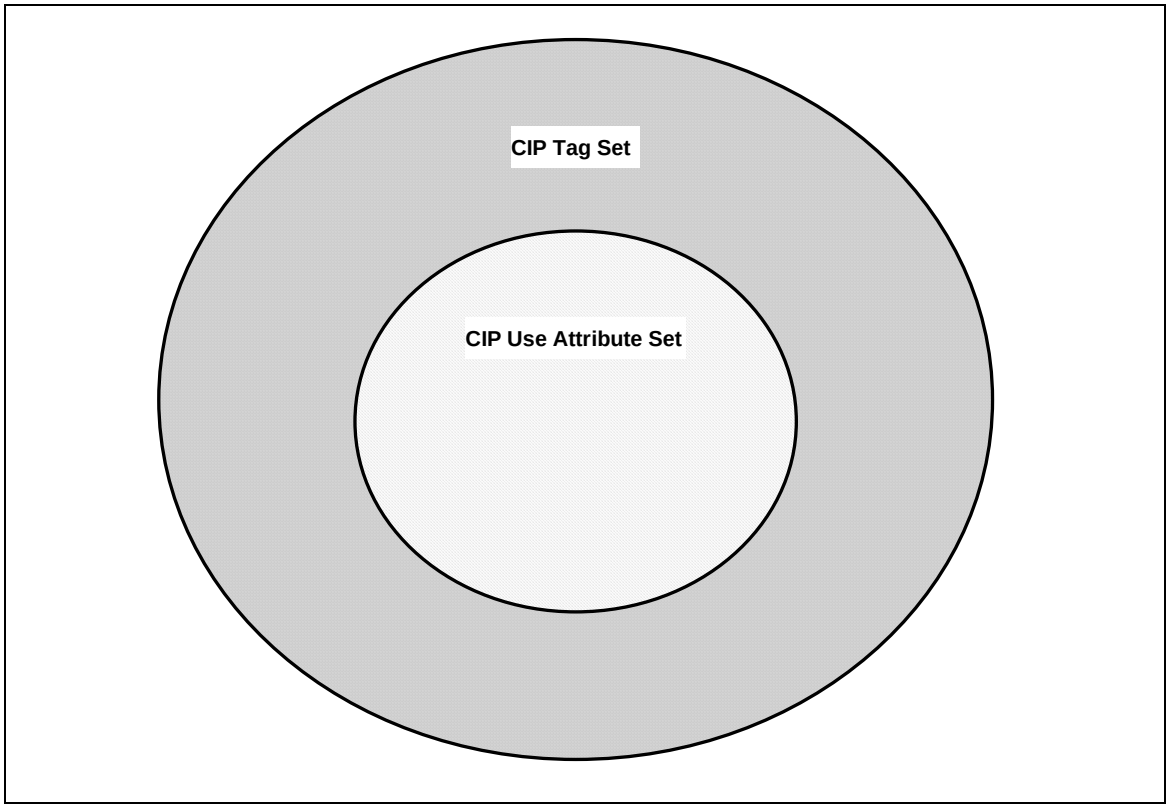

Figure 3-4: CIP Tags and Attribute Sets

The *schema elements* defined in the *database schemas* are mapped to physical data fields at each *target* site. Within a site, the data fields may have different names than those defined in the *schemas*. This is reasonable and valid as long as the *target* is suitably constructed to be able to map the CIP *schema elements* to the locally defined *elements*. Also, it is possible that only a subset of *schema elements* from the *database schema* are available at a particular site (or that a standard *Use attribute* is not indexed at a particular site). These cases are not constrained by the CIP specification.

For instance, if a mandatory CIP *schema element* is not supported at a particular site, the Retrieval Manager would still return this *schema element* to the client when requested, but it would return it with a NULL value so as to identify that the *element* is not recognised at the site (i.e. the *element* is recognised to be a valid CIP *element*, but its value is 'unknown').

In addition to the CIP *elements* each catalogue site may have additional local *elements*. The same principle applies as for *Use attributes*. Again, as long as the names are defined within the *Explain database*, a client may query those *attributes*, or display the *elements*, although they will be meaningful for local queries only, not for queries distributed to remote Retrieval Managers.

[17] Two facts are worth stressing. Firstly, it is quite natural that the set of *Use attributes* are a subset of the set of *elements* used to describe the *schemas*, since usually the search criteria used in a search correspond to the items which must be retrieved, but this does not necessarily have to be so. Secondly, it is also natural that not all the *elements* used to describe the *schemas* are used as *Use attributes*, because some of those *elements* are not appropriate as search criteria (i.e. a browse image).

3.5.3.3 Retrieval Record

3.5.3.3.1 Retrieval Record Determination

A *retrieval record*, which is identified by an entry in a *result set*, can be considered as the subset of a *database record* which can be transferred back to the client as a result of a *retrieval request*.

To determine the contents of a *retrieval record* the process demonstrated in Figure 3-5 is performed:

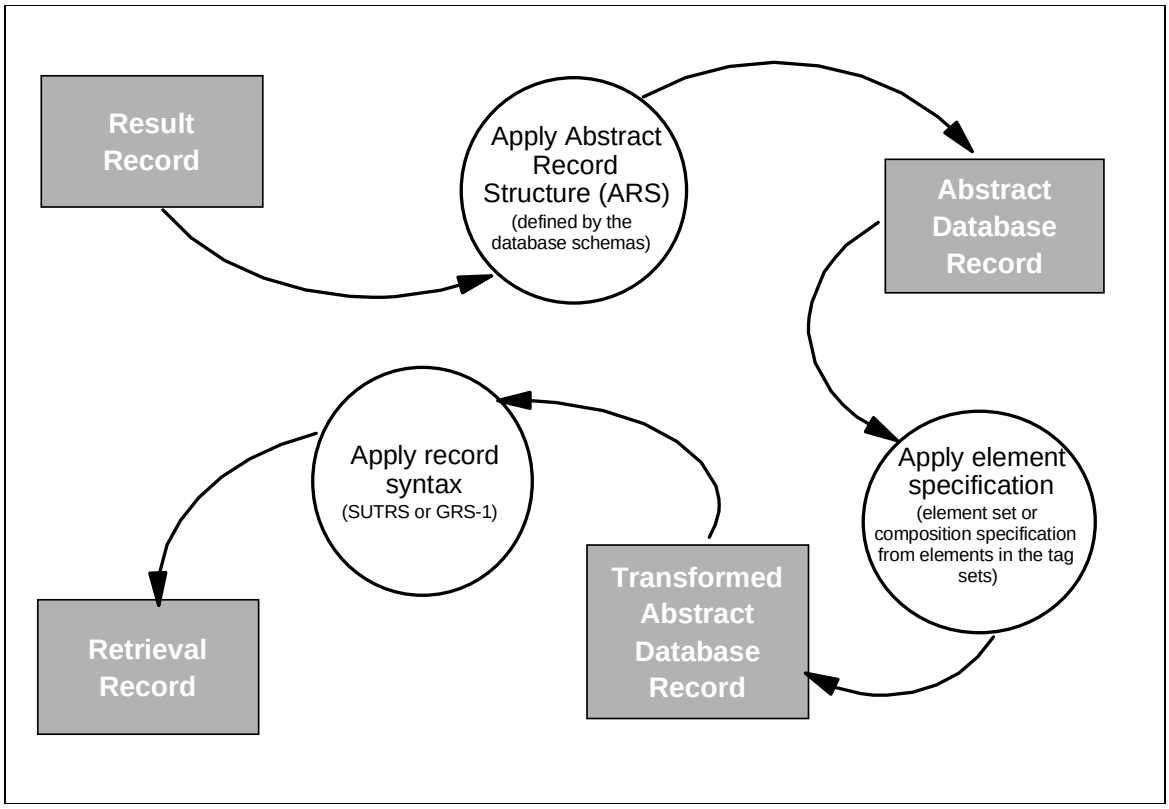

Figure 3-5: Record Retrieval Process

This is explained by the following steps:

1. The Retrieval Manager applies the *abstract record structure* (ARS) defined by the *database schema* for the *database* to which the *database record* belongs, yielding an *abstract database record*.
2. The Retrieval Manager applies a *record composition specification* to the *abstract database record*. A *record composition specification* is composed of *element requests*, each specifying an *element*. The transformation which is applied can be thought as a filter, where all the *elements* requested in the *record composition specification* are selected from the *abstract database record*. The transformation therefore yields a subset of the *abstract database record*. Note that a null transformation is possible (i.e. all the *elements* are selected).
3. The Retrieval Manager applies a *record syntax* to the filtered *abstract database record*, yielding the *retrieval record*.

For example, assuming that the *abstract record structure* for the Product *schema* (as defined in Appendix C.3) is applied and that an *element composition* requiring the schema elements 'TemporalCoverage', 'SpatialCoverage', and 'ArchivingCentreId' is applied and that the *GRS-1 syntax* is applied, the resulting *retrieval record* could be represented as shown in Figure 3-6:

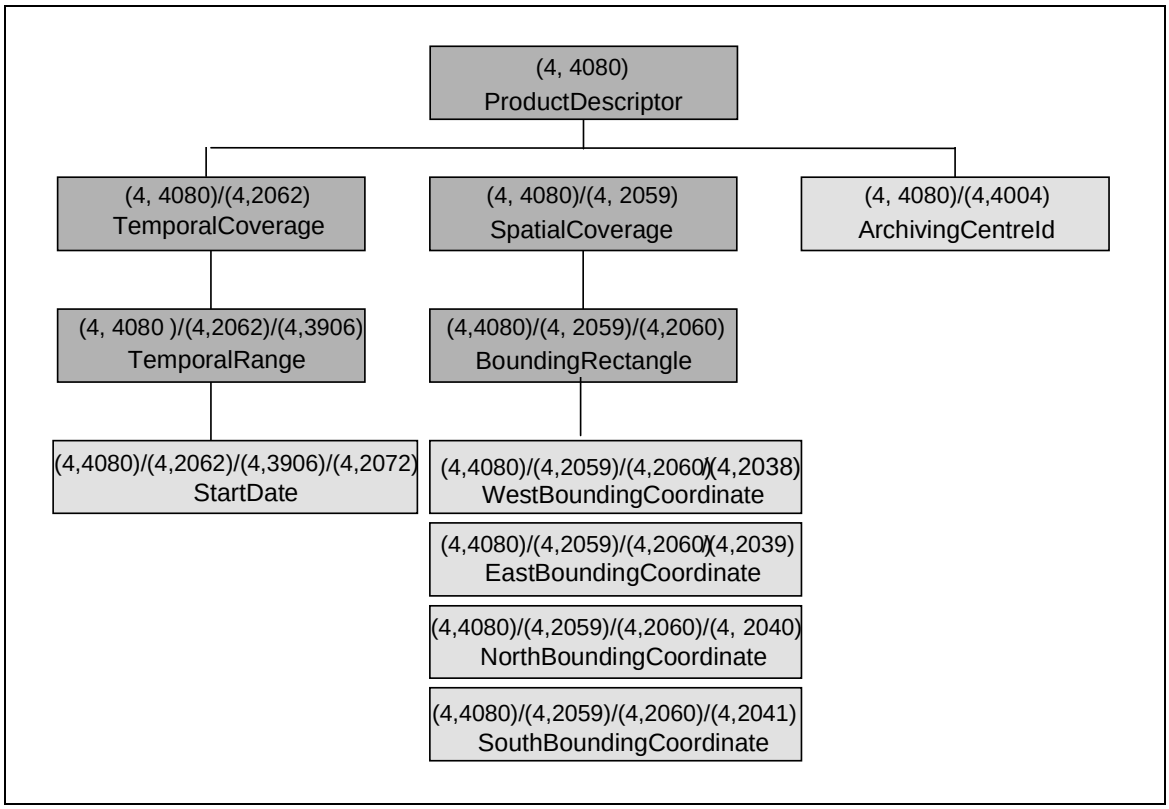

Figure 3-6: Retrieval Record

3.5.3.3.2 Element Set Names

The CIP supports eight defined *element set names* for retrieval. The 'Full' and 'Brief' names are provided as required defaults (i.e. each Retrieval Manager must support these two) for compatibility with clients possessing minimal capabilities. In addition, a 'CIP Full' *element set name* is provided for retrieval of the CIP standard *elements* in case a custom *database schema* is used (instead of the CIP *database schema*). 'Summary' *element set name* is mainly used for interoperability with Z39.50-Version 2 profiles. A 'Browse' *element set name* is provided for retrieval of browse data, the 'Opt' *element set name* is provided for the retrieval of options For the retrieval of the local attributes, the 'Local Attributes' *element set name* is defined. Finally a 'Collection Member' *element set name* is provided for retrieval of collection hierarchy information. These eight *element set names* are described below; their element contents are defined in Appendix C.4.

- **Full** (F): the Full *element set* includes all defined standard *elements* from the appropriate *database schema* (for the CIP, this is usually the CIP *database schema*, as defined in Appendix C.3), and so, when applied, results in a null transformation. This is a large set of *elements*, but it ensures that clients receive everything their users may need to evaluate the *retrieval record* for further processing. Note that, whilst all schema *elements* are returned, some *elements* may be meaningless for the record that is actually returned, and may contain undefined values.
- **CIP Full** (CF): the CIP Full *element set* includes all defined standard CIP *elements* from the CIP *database schema* as defined in Appendix C.3. When the CIP *database schema* is used, the Full and CIP Full element sets are therefore equivalent. However, when a custom *database schema* is used (i.e. a custom *schema* defined by a data provider and containing local attributes in addition to the standard CIP ones), the CIP Full *element set* contains uniquely the standard CIP *elements*, whereas the Full *element set* contains **all** the *elements* defined in the custom *database schema*, i.e. the standard CIP *elements* and the custom local *elements*.

- **Brief** (B): the Brief *element set* includes a minimal subset of the defined standard *schema elements* available from the appropriate *database schema*.
- **Summary** (Sum): the Summary *element set* includes a subset of the defined standard *schema elements* that is appropriate for interoperability with the GEO profile[GEO].
- **Browse** (Br): the Browse *element set* includes a subset of the defined *schema elements* which are appropriate for retrieval of browse data alone.
- **Options** (Opt): the Options *element set* includes a subset of the defined *schema elements* which are appropriate for retrieval of options alone.
- **Local Attributes** (LA): the Local Attributes *element set* includes a subset of the defined standard *schema elements* that is appropriate for retrieval of the local attributes in a product descriptor.
- **Collection Members** (CM): the Collection Members *element set* includes a subset of the defined *schema elements* which are appropriate for retrieval of the collection hierarchy tree.

Each individual Retrieval Manager may also wish to define other *element set names* which are applicable to its local site. For instance, each Retrieval Manager could define a 'Local Brief' *element set name* in order to provide a default for retrieval of *brief records* of its local data. They may also define *element set names* for specific item types available through the local server. At initialisation, such additional *element set names* can be provided to clients through the *Explain database*; there is no need to define them a priori in this specification.

The content of the *retrieval record*, which results from the application of the various *element sets* described above on the CIP *database schema*, is described in Appendix C.3.

3.5.3.3.3 Composition Specifications

As an alternative or addition to the use of predefined *element set names* to define desired *elements* for retrieval, the client can specify, in the *present request*, a list of *elements* to be retrieved. This capability is referred to as a *composition specification*. Due to the disparity of product descriptor contents and the large number of *elements* available in the EO environment, it is expected that this will be a common approach taken by experience users via their clients.

In a *composition specification*, the client specifies the set of *elements* that it wants to be retrieved from each record in the *result set* by defining a set of *element requests* (i.e. a separate *request* for each *element* to be retrieved). This enables full customisation of element retrieval by selecting, one by one, the desired *elements*[18]. Note, however, that the client can use a predefined *element set* in a *composition specification* (where each *element* in the *element set* is treated as an individual item in the *request*). This allows, for instance, to use a predefined *element set* (i.e. one of the CIP pre-defined *element sets*) and customise it by adding other *elements* (e.g. local *elements*).

As a practical matter, for users and clients who wish to view a different set of *database elements* than those defined in the standard *element names*, there are alternative implementation approaches which can be used:

1. The client can request the Full *element set* and customise that list locally to suit the user, or
2. The client can obtain from the user a list of desired *elements* which it can then pass to the Retrieval Manager through a *composition specification*.

Each approach has the functionality to return similar results to the user, although the latter approach has the additional capability of providing retrieval of individually specified local *elements*, i.e.

[18] Note that if the *element* selected is a *composite element* (also called *compound element*) the whole *composite* is selected (i.e. the *composite element* and all its *sub-elements*).

performing the filtering at the Retrieval Manager rather than at the client and hence reduce network traffic.

3.5.3.3.4 Record Syntax

Each record to be returned in a *present response*, or *piggybacked* in a *search response*, can be thought of as an *abstract database record* consisting of *element* names and values. The *target* applies a *record syntax* to each *abstract database record* before returning it to the *origin*. ***GRS-1*** is the preferred syntax for CIP due to its rich structure and provisions for data manipulation by the client. The ***SUTRS*** *record syntax* in HTML is the syntax that is used when interoperability with a *Z39.50*-Version 2 profile is desired.
The *record syntaxes* supported by CIP are the following (each syntax is formally defined in Section 3.5.3.5.4):

- ***SUTRS*** *record syntax*: in SUTRS, the formatting of the record contents is handled by the *target*, and the client receives a record devoid of structure. With this syntax the *origin* may display the records with little or no analysis or manipulation. The record consists simply of a string of textual data which is not well suited to automated parsing by the client, but may be convenient for simple user displays.
- ***GRS-1*** *record syntax*: in *GRS-1* the record, whose structure is defined by the *record syntax*, is passed from the *target* to a client, and the client software has more flexibility in processing the record contents for display.
- ***EXPLAIN*** *record syntax*: a *present request* for *Explain records* must specify the *Explain record syntax* as the preferred *record syntax*. Each *Explain information category* has its own record layout, these are defined in the Z39.50 specification[Z3950].
- ***ES TaskPackage*** *record syntax*: a *present request* for *Extended Services Task Package records* must specify the *ES Task Package record syntax* as *record syntax*. Each specific *Extended Service* customises the record delivered by providing task specific parameters within the record. The record syntax and the task specific parameters are presented in Section 3.5.7.

3.5.3.4 Records Segmentation

The CIP supports *Z39.50 level 2 segmentation*. This means that, in addition to being able to segment an *aggregate present response* into multiple *segments* containing one or more integral *retrieval record(s)* (i.e. *level 1 segmentation*), the *target* may fragment the *retrieval records* and thus retrieve *retrieval records* that may span *segments*.

From the *segmentation* point of view, a *retrieval record* is considered as a string of bytes, and a *fragment* is defined as a sub-string of a *retrieval record*[19]. In the procedure for *level 2 segmentation*, three kinds of *fragments* are distinguished:

- *Starting fragment*, defined as a *fragment* that starts at the beginning of a record.
- *Intermediate fragment*, defined as a *fragment* that neither starts at the beginning nor ends at the end of a record.
- *Final fragment*, defined as a *fragment* that ends at the end of a record.

[19] Note that with this definition an integral record is **not** a fragment.

The following procedure applies for *level 2 segmentation*:

- If all the *retrieval records* fit into a *segment*, the *segment* is sent to the *origin* as a *Present response*, thus resulting in a *simple present response*.
- Otherwise, the following process is performed:
  - Step 1:

    The *target* fits as many integral *retrieval records* as possible into the first *segment* and, if possible, the *target* fits a *starting fragment* of the following *retrieval record* in the space remaining within that *segment*. The *segment* is then sent as a *Segment request*.
  - Step 2:
    - If possible, the *target* fits the remaining of the fragmented *retrieval record* into the next *segment* and then fills as many integral *retrieval records* as possible within the space remaining within that *segment*. If the last of the requested *retrieval records* is placed in the *segment*, the *segment* is sent as a *Present response*. Otherwise, the *segment* is sent as a *Segment request* and then Step 2 is reapplied until the last of the requested *retrieval records* is place in a *segment*.
    - Otherwise, the *target* sends as many *Segment requests* containing *intermediate fragments* as necessary. It then restarts Step 2 in order to fits the *final fragment* into the beginning of the following *segment*.

The process for *level 2 segmentation* is illustrated in Figure 3-7.

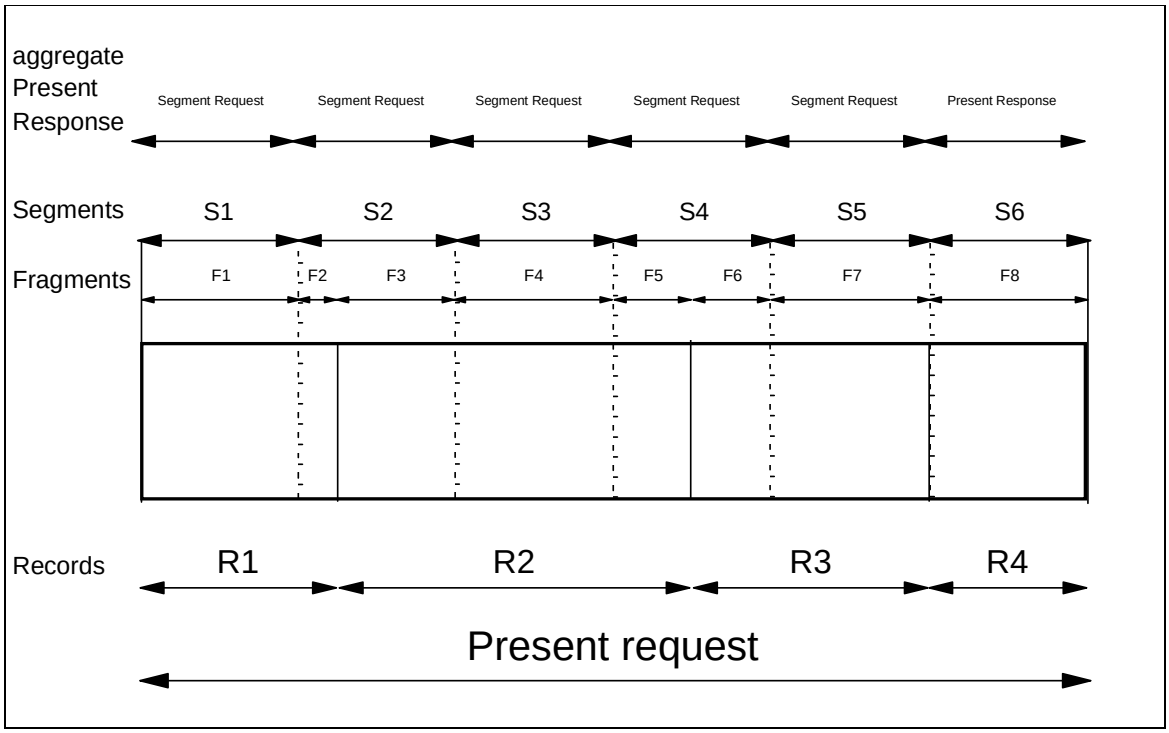

Figure 3-7: Segmentation and Fragmentation

3.5.3.5 Retrieval Service Formal Definitions

The *Retrieval facility* consists of two services: the *Present service* and the *Segment service*, which are presented in the subsections below.

3.5.3.5.1 Present Request

Through the *present request*, the client can fully specify (on behalf of the user) all the characteristics of the records to be returned as a result of a completed search. This can be done either through a pre-defined named *element set* (an *element set name*) or by specifying *element requests* which define a list of elements and their order of presentation. *Present requests* can be submitted iteratively one or more times, obtaining a set of records whose size is defined by the client in each *present request*, until all desired records have been returned.

The *present request* is structured as describe in Table 3-10. A semantic definition of each required and optional part is provided adjacent to the appropriate formal ASN.1 definition.

Table 3-10: Present Request Usage

ASN.1 Definition	Meaning

`PresentRequest ::= SEQUENCE{`
`referenceId ReferenceId ~~OPTIONAL~~,`
`resultSetId ResultSetId,`
`resultSetStartPoint [30] IMPLICIT INTEGER,`
`numberOfRecordsRequested [29] IMPLICIT INTEGER,`
`additionalRanges [212] IMPLICIT SEQUENCE OF Range OPTIONAL,`
`-- additionalRanges may be included only if version 3 is in force.`
`recordComposition CHOICE{`
`simple [19] ElementSetNames,`
`complex [209] IMPLICIT CompSpec}`
`OPTIONAL,`
`preferredRecordSyntax [104] IMPLICIT OBJECT IDENTIFIER OPTIONAL,`
`maxSegmentCount [204] IMPLICIT INTEGER OPTIONAL, -- level 1 or 2`
`maxRecordSize [206] IMPLICIT INTEGER OPTIONAL, -- level 2 only`
`maxSegmentSize [207] IMPLICIT INTEGER OPTIONAL, -- level 2 only`
`otherInfo OtherInformation OPTIONAL}`
	A **PresentRequest** is used by the origin to request retrieval records according to their position within a result set maintained by the target. It consists of the following elements:
	• **referenceId**, which is a reference identifier assigned to the operation (consisting of a PresentRequest and a PresentResponse) initiated by the PresentRequest and which allows to match a request with a response.
	• **resultSetId**, which is the identifier of the result set from which records are to be retrieved.
	• **resultSetStartPoint**, which is the relative position within the result set from which retrieval starts.
	When collections are searched, **resultSetStartPoint** is calculated as if the result set was a flat set of records formed by the flattening of the nested internal structure of the result set (which reflects the hierarchical structure of the collection searched), thus ignoring any information about the internal structure of the result set.
	• **numberOfRecordsRequested**, which is the number of records which must be retrieved from the result set starting at the position defined by **ResultSetStartPoint**.
	When collections are searched, **numberOfRecordsRequested** is calculated as if the result set was a flat set of records, thus ignoring any information about the internal structure of the result set. **numberOfRecordsRequested** therefore corresponds to the number of records at the leaf level of the result set that are

ASN.1 Definition	Meaning
	requested to be retrieved. • **additionalRange**, which allows to require additional, non-contiguous ranges in the result set to be retrieved. • **recordComposition**, which specifies the desired composition of retrieved records. A **recordComposition** is either simple or complex. A simple **recordComposition** is represented by **ElementSetNames**, each identifying a set of elements. A complex RecordComposition is represented by a **CompSpec**, which is a composition specification. • **preferredRecordSyntax**, which is the OID of the syntax that must be used when a record is retrieved. • Segmentation information, which consists of **maxSegmentCount**, which specifies the maximum number of segments that the target may include in an aggregate **PresentResponse**, **maxSegmentSize**, which is the size of the largest allowable segment and **maxRecordSize**, which is the size of the largest allowable retrieval record. • **otherInfo**, which may be used for additional information not specified by the standard.

`Range ::= SEQUENCE{`
`  startingPosition  [1] IMPLICIT INTEGER,`
`  numberOfRecords  [2] IMPLICIT INTEGER}`
	Range consists of a **startingPosition**, which is the relative position within the result set from which retrieval starts, and a **numberOfRecords**, which is the number of records which must be retrieved from the result set starting at the position defined by startingPosition.

`ElementSetNames ::= CHOICE {`
`  genericElementSetName  [0] IMPLICIT InternationalString,`
`  databaseSpecific  [1] IMPLICIT SEQUENCE OF SEQUENCE{`
`  dbName DatabaseName,`
`  esn ElementSetName}}`
	ElementSetNames are composed of either a generic **ElementSetName** or a list of database specific **ElementSetNames**.
	For the CIP, a generic **ElementSetName** is one of the following: 'F' (Full), 'CIP' (CIP Full), 'B' (Brief), 'Sum' (Summary), 'Br' (Browse), 'Opt' (Options), 'LA' (Local Attributes), or 'CM' (Collection Member).
	Database specific **ElementSetNames**, which are used for element set names locally defined by a Retrieval Manager, are identified by the name of the database **DatabaseName** and the **ElementSetName** itself.

`ElementSetName ::= [103] IMPLICIT InternationalString`	An **ElementSetName** identifies a set of elements indicating the desired composition of a retrieved record.

`CompSpec ::= SEQUENCE{`
`  selectAlternativeSyntax  [1] IMPLICIT BOOLEAN,`
`  -- See comment for recordSyntax, below.`
`  generic  [2] IMPLICIT Specification OPTIONAL,`
`  dbSpecific  [3] IMPLICIT SEQUENCE OF SEQUENCE{`
`  db [1] DatabaseName,`
`  spec [2] IMPLICIT Specification} OPTIONAL,`
`  recordSyntax  [4] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER`
`  OPTIONAL}`
	A **CompSpec** is a composition specification. It is composed of the following elements:
	• A **selectAlternativeSyntax** flag, which indicates if the target may select an alternative syntax in case none of the syntaxes provided in **RecordSyntax** are supported by the target.
	• A **Specification**, which specifies the selection of elements which are part of the composition specification. This **Specification** may be either **generic**,

ASN.1 Definition	Meaning
	specifying a selection of CIP elements, **database specific**, specifying a selection of database specific elements, or both. • A **recordSyntax**, which specifies the list of record syntaxes that may be used for retrieval. For each record, the target selects the first record syntax in this list that it can support. If the list is exhausted, the target may select an alternative syntax if **SelectAlternativeSyntax** is 'true'. The GRS-1 record syntax is recommended for use within CIP.

`Specification ::= SEQUENCE{`
`  schema [1] IMPLICIT OBJECT IDENTIFIER OPTIONAL,`
`  elementSpec [2] CHOICE{`
`  elementSetName [1] IMPLICIT InternationalString,`
`  externalEspec [2] IMPLICIT EXTERNAL} OPTIONAL}`
	A **Specification** is defined by identifying the **schema** used (if it is not the default **schema**) and by specifying an **elementSpec**, which is the specification of the elements from the **schema** which are selected. **elementSpec** is either identified by an **elementSetName**, or it is an external specification **externalEspec**. For the CIP, the external specification used shall be **Espec-1**.

`Espec-1 ::= SEQUENCE{`
`  elementSetNames [1] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,`
`  defaultVariantSetId [2] IMPLICIT OBJECT IDENTIFIER OPTIONAL,`
`  defaultVariantRequest [3] IMPLICIT Variant OPTIONAL,`
`  defaultTagType [4] IMPLICIT INTEGER OPTIONAL,`
`  elements [5] IMPLICIT SEQUENCE OF ElementRequest OPTIONAL}`
	An external specification defined according to **Espec-1** identifies a set of elements to be selected in the form of a set of **ElementRequest**. It contains the following:
	• **elementSetNames**, each identifying a predefined set of elements. Each element in the element set is to be treated as an **ElementRequest** in the form of **SimpleElement**, where **Occurrence** is 1.
	• A default variant set identifier **defaultVariantSetId** which applies whenever a **VariantRequest** does not include a **VariantSedId**.
	• A default variant request **Variant**, which is the default variant to be applied to **ElementRequest** (if variants are requested).
	• A **defaultTagType**, which is the tag type applying whenever a tag type is omitted within the tag path of a tag.
	• A list of **elements** specified as a list of **ElementRequests**.

`ElementRequest ::= CHOICE{`
`  simpleElement [1] IMPLICIT SimpleElement,`
`  compositeElement [2] IMPLICIT SEQUENCE{`
`  elementList [1] CHOICE{`
`  primitives [1] IMPLICIT SEQUENCE OF InternationalString,`
`  specs [2] IMPLICIT SEQUENCE OF SimpleElement},`
`  deliveryTag [2] IMPLICIT TagPath,`
`  variantRequest [3] IMPLICIT Variant OPTIONAL}}`
	An **ElementRequest** is either a **simpleElement** (i.e. an atomic element) or a **compositeElement**.
	A **compositeElement** contains:
	• An **elementList**, which is the list of elements composing the **compositeElement**. The **elementList** contains either a list of **primitives**, which are sets of elements identified by **ElementSetNames** and whose union composes the **CompositeElement**, or list of **SimpleElements**.
	• A **deliveryTag**, which is the **TagPath** for the **compositeElement**. It may not contain wild cards.
	• A **variantRequest** identifying the applicable **Variant**.

ASN.1 Definition	Meaning

`SimpleElement ::= SEQUENCE{`
`  path  [1] IMPLICIT TagPath,`
`  variantRequest  [2] IMPLICIT Variant OPTIONAL}`
	A **SimpleElement** consists of:
	• A **TagPath** identifying the element. The elements defined for the CIP are described in Appendix B.
	• A **VariantRequest** identified by a **Variant**.

`TagPath ::= SEQUENCE OF CHOICE{`
`  specificTag  [1] IMPLICIT SEQUENCE{`
`  tagType  [1] IMPLICIT INTEGER OPTIONAL,`
`  tagValue  [2] StringOrNumeric,`
`  occurrence [3] Occurrences OPTIONAL`
`  -- default is 'first occurrence'`
`  },`
`  wildThing  [2] Occurrences,`
`  wildPath  [3] IMPLICIT NULL`
`  }`
	A **TagPath** consists in a list of tags, each represented by one of the following:
	• A **specificTag**, which consists of a **tagType** (identifying the TagSet from which the Tag is selected; if omitted, the **DefaultTagType** or, if missing, the default listed in the schema, applies), a **TagValue** (uniquely identifying the Tag within its TagSet) and an **Occurrence**, which allows to select a specific occurrence of a particular Tag (this is meaningful for recurring elements only).
	• A **wildThing** or **wildPath**, which allows the use of wild cards in the selection of elements, either in the specification of an occurrence (**wildThing**) or in the specification of a tag path (**wildPath**).
	The use of wild cards is fully explained in [Z3950], Appendix RET 3.1.1.4.

`Occurrences ::= CHOICE{`
`  all  [1] IMPLICIT NULL,`
`  last  [2] IMPLICIT NULL,`
`  values  [3] IMPLICIT SEQUENCE{`
`  start  [1] IMPLICIT INTEGER,`
`  howMany  [2] IMPLICIT INTEGER OPTIONAL`
`  }}`
	Occurrences allows to specify the desired occurrences of an element within a record. The following **Occurrences** can be selected:
	• **all** the allows to retrieve all occurrences of an element.
	• **last** allows to retrieve the last occurrence of an element within a record.
	• **values** allows to specify a range of occurrences by specifying the **start** occurrence and number **howMany** of occurrences from that starting position.

3.5.3.5.2 Segment Request

When the records requested by a *present request* will not fit in a single *segment*, the *target* returns multiple *segments*, each containing a portion of the requested records. Every *segment* except for the last, which is returned as a *present response*, is returned as a *segment request*. Note, because of modelling constraints, the *segment service* is modelled as a *request* (and not a *response*), although it is actually used as a *response* to a *present request*.

Table 3-11: Segment Request Usage

ASN.1 Definition	Meaning
`segmentRequest [45] IMPLICIT Segment`	A **segmentRequest** consists of a **Segment**.

`Segment ::= SEQUENCE{`
`-- Segment PDU may only be used when version 3 is in force,`
`-- and only when segmentation is in effect.`
`referenceId ReferenceId ~~OPTIONAL~~,`
`numberOfRecordsReturned [24] IMPLICIT INTEGER,`
`segmentRecords [0] IMPLICIT SEQUENCE OF NamePlusRecord,`
`otherInfo OtherInformation OPTIONAL}`
	A **SegmentRequest** returns the records, or portion of records, requested during a **PresentRequest**. A **Segment** contains the following information:
	• The **referenceId**, which is the reference identifier of the operation initiated by the **PresentRequest**.
	• **numberOfRecordsReturned**, which is the total number of response records and starting fragments included within the segment.
	• **segmentRecords**, which contains response records and/or fragments of records.
	• **otherInfo** about the **SegmentRequest**.

3.5.3.5.3 Present Response

The *present response* is structured as follows. A semantic definition of each required and optional part is provided adjacent to the appropriate formal ASN.1 definition.

Table 3-12: Present Response Usage

ASN.1 Definition	Meaning

`PresentResponse ::= SEQUENCE{`
`referenceId ReferenceId ~~OPTIONAL~~,`
`numberOfRecordsReturned [24] IMPLICIT INTEGER,`
`nextResultSetPosition [25] IMPLICIT INTEGER,`
`presentStatus PresentStatus,`
`records Records OPTIONAL,`
`otherInfo OtherInformation OPTIONAL}`
	A **PresentResponse** returns the records requested during a **PresentRequest**. It contains the following information:
	• A **referenceId**, which is the reference identifier of the operation initiated by the **PresentRequest**.
	• **numberOfRecordsReturned**, which is the total number of records returned by the **PresentResponse**.
	• The **nextResultSetPosition**, which is the next position in the result set from the position of the last record returned in the response.
	• The **presentStatus**, which is the status of the **PresentResponse**, indicating if it was successful or not.
	• The **records** returned.
	• **otherInfo** about the PresentResponse.

ASN.1 Definition	Meaning

`PresentStatus ::= [27] IMPLICIT INTEGER{`
`success (0),`
`partial-1 (1),`
`partial-2 (2),`
`partial-3 (3),`
`partial-4 (4),`
`failure (5)}`
	The **PresentStatus** indicates the status of the **PresentResponse**. The response is either a success, a partial success (not all of the expected response can be returned) or a failure. The possible status are the following:
	• **success**, which indicates that all the expected response records are available.
	• **partial-1**, which indicates that the request was terminated by an access control.
	• **partial-2**, which indicates some records will not fit within the preferred message size.
	• **partial-3**, which indicates that the request was terminated by resource control at origin request.
	• **partial-4**, which indicates that the request was terminated by resource control by the target.
	• **failure**, which indicates that none of the expected response records can be returned. One or more non-surrogate diagnostic records is returned.

`Records ::= CHOICE{`
`responseRecords [28] IMPLICIT SEQUENCE OF NamePlusRecord,`
`nonSurrogateDiagnostic [130] IMPLICIT DefaultDiagFormat,`
`multipleNonSurDiagnostics [205] IMPLICIT SEQUENCE OF DiagRec}`
	The **Records** returned are one of the following:
	• A list of **responseRecords**.
	• A single diagnostic record in the **defaultDiagFormat**.
	• Multiple diagnostic records **diagRec**.

`NamePlusRecord ::= SEQUENCE{`
`name [0] IMPLICIT DatabaseName ~~OPTIONAL~~,`
`record [1] CHOICE{`
`retrievalRecord [1] EXTERNAL,`
`surrogateDiagnostic [2] DiagRec,`
`startingFragment [3] FragmentSyntax,`
`intermediateFragment [4] FragmentSyntax,`
`finalFragment [5] FragmentSyntax}}`
	NamePlusRecord contains a **DatabaseName** (for details see Appendix E.7), identifying the database from which the record is retrieved, and the **Record** itself.
	The **Record** can be one of the following:
	• A **retrievalRecord**, which is defined externally.
	• A **surrogateDiagnostic** record.
	• A **fragments** of a record. The following fragments are identified:
	  • **startingFragment**, which is a fragment that starts at the beginning of a record.
	  • **intermediateFragment**, which is a fragment that neither starts at the beginning nor ends at the end of a record.
	  • **finalFragment**, which is a fragment that ends at the end of a record.

`DiagRec ::= CHOICE{`
`defaultFormat DefaultDiagFormat,`
`externallyDefined EXTERNAL}`
	A diagnostic record **DiagRec** is either a record formatted according to the default diagnostic format **DefaultDiagFormat** or a record formatted according to an **externallyDefined** format.

ASN.1 Definition	Meaning

DefaultDiagFormat::= SEQUENCE{
diagnosticSetId OBJECT IDENTIFIER,
condition INTEGER,
addinfo CHOICE{
v2Addinfo VisibleString, -- version 2
v3Addinfo InternationalString -- version 3
}}
	The **DefaultDiagFormat**, which is the default diagnostic format, consists of the diagnostic set identifier **diagnosticSetId**, a **condition** and version dependent additional information **addInfo**.

FragmentSyntax ::= CHOICE{ externallyTagged EXTERNAL, notExternallyTagged OCTET STRING}	The **FragmentSyntax** contains the fragment of a record and, if required, the syntax used for the fragment (see Table 3-13 for more detail). **FragmentSyntax** is either: • **externallyTagged**, in which case the syntax which would have been used if the record were not segmented is specified with the actual fragment. In this case, the syntax used for the initial fragment must be used for all the subsequent fragments (i.e. intermediate and final fragments) of the record. or • **notExternallyTagged**, in which case only the actual fragment is included (i.e. no additional syntactical information is included).

3.5.3.5.4 Record Syntaxes

The ***Fragment Syntax***, defined by OID {Z39-50-recordSyntax fragment}, is defined below:

Table 3-13: Fragment Syntax Usage

ASN.1 Definition	Meaning
Fragment ::= SEQUENCE { realSyntax [1] IMPLICIT OBJECT IDENTIFIER OPTIONAL, -- Required in starting fragment. -- Must be omitted in subsequent fragments. remainingOctets [2] IMPLICIT INTEGER OPTIONAL, -- Target estimate of how many octets -- left after this fragment. fragment [3] IMPLICIT OCTET STRING}	The **Fragment** syntax is used to reflect the real syntax (via its OID) that would have otherwise been used if the record were not segmented. The **Fragment** syntax consists of the following information: • **realSyntax** contains the OID of the syntax that would have been used if the record were not segmented. The **realSyntax** must be specified in the starting fragment and only in the starting fragment (i.e. it must be omitted in the intermediate fragments and in the final fragment). • **remainingOctets** contains the target's estimate of how many octets are left after the current fragment. • **fragment** contains the actual fragment.

The ***SUTRS*** *record syntax*, defined by OID {Z39-50-recordSyntax SUTRS}, can be available in two variants:

- **HTML** formatted MIME-type records: This is the default ***SUTRS*** *record syntax*. The HTML records are generated from the SGML Document Type Declaration (DTD) that is defined in Appendix D. This variant shall be always returned as response to a *Present request* originated from a Z39.50-Version 2 profile. The format must reflect the *schema* structure, i.e.:

  Full_Compound_Element_Name : Value

  Compound_Element_Name: Value

  ....

  Full_Leaf_Element_Name: Value

  Grouped *elements* are indented. e.g.

  Full_Leaf_Element_Name : Value

- **text**: where the string is a sequence of *element names* followed by an equals sign ('='), followed by the *element value* terminated with a semi-colon (';'). *Origins* must not depend on being able to parse the records in order to change the presentation format.

Table 3-14: SUTRS Record Syntax Usage

ASN.1 Definition	Meaning
`SutrsRecord ::= InternationalString`	A **SUTRSRecord** is a simple unstructured string. The recommended maximum line length is 72 characters.

The ***GRS-1*** *record syntax*, defined by OID {Z39-50-recordSyntax GRS-1}, is defined below:

Table 3-15: GRS-1 Record Syntax Usage

ASN.1 Definition	Meaning
`GenericRecord ::= SEQUENCE OF TaggedElement`	A **GenericRecord** is a list of **TaggedElement**. The CIP **TagElements** are defined in Appendix B.

`TaggedElement ::= SEQUENCE {`
`  tagType [1] IMPLICIT INTEGER OPTIONAL,`
`  tagValue [2] StringOrNumeric,`
`  tagOccurrence [3] IMPLICIT INTEGER OPTIONAL,`
`  content [4] ElementData,`
`  metaData [5] IMPLICIT ElementMetaData OPTIONAL,`
`  appliedVariant [6] IMPLICIT Variant OPTIONAL}`
	A **TagElement** is an element defined in a tag set. Each element consists of the following information:
	• A **tagType**, which identifies the tag set in which the element belongs.
	• A **tagValue**, which uniquely identifies the element within the tag set.
	• A **tagOccurrence**, which indicates the occurrence of the element within the

ASN.1 Definition	Meaning
	database record. • The **content** of the element, which is the actual content of the element within the **GenericRecord**. • **metaData**, which contains metadata information about the element. • **appliedVariant**, which contains the **Variant** applied to the element.

`ElementData ::= CHOICE{`
`  octets  OCTET STRING,`
`  numeric  INTEGER,`
`  date  GeneralizedTime,`
`  ext  EXTERNAL,`
`  string  InternationalString,`
`  trueOrFalse  BOOLEAN,`
`  oid  OBJECT IDENTIFIER,`
`  intUnit [1] IMPLICIT IntUnit,`
`  elementNotThere [2] IMPLICIT NULL, -- element requested but not there`
`  elementEmpty [3] IMPLICIT NULL, -- element there, but empty`
`  noDataRequested [4] IMPLICIT NULL, -- variant request said 'no data'`
`  diagnostic [5] IMPLICIT EXTERNAL,`
`  subtree [6] SEQUENCE OF TaggedElement`
`  }  -- recursive, for nested tags`
	ElementData contains the actual value of the **Element**.
	The content of the **ElementData** is one of the following:
	• string of octets;
	• numeric value;
	• date;
	• value whose type is externally defined;
	• string;
	• boolean value;
	• object identifier;
	• integer and its unit;
	• indication that an element requested is not there;
	• indication that an element is there but is empty;
	• indication that no data was requested (and therefore none is provided);
	• externally defined diagnostic;
	• sub-tree of **TagElements** (recursive definition for nested tags).

`IntUnit ::= SEQUENCE{`
`  value  [1] IMPLICIT INTEGER,`
`  unitUsed  [2] IMPLICIT Unit}`
	An **IntUnit** consists of an integer **value** and the **Unit** used for this **value**.

`Unit ::= SEQUENCE{`
`  unitSystem  [1] InternationalString ~~OPTIONAL~~,  -- e.g. 'SI'`
`  unitType  [2] StringOrNumeric OPTIONAL,  -- e.g. 'mass'`
`  unit  [3] StringOrNumeric ~~OPTIONAL~~,  -- e.g. 'kilograms'`
`  scaleFactor  [4] IMPLICIT INTEGER OPTIONAL  -- e.g. 9 means 10**9`
	A **Unit** is defined by the **unitSystem** which defines the **Unit**, the **unitType** of the **Unit**, the **unit** itself and a **scaleFactor**.

3.5.4 Result-set delete Facility

The *Result Set Delete facility* consists of a single service, the *Delete service*, which allows a *target* to request an *origin* to delete specified result sets created during a Z-association[20].

The *Delete Result Set operation* must be performed in accordance to the *Search operation* which originally created the *result set* to be deleted so that all the results which were created by the original *Search operation* are indeed deleted by the *Delete Result Set operation*. This means that if the *result set* to be deleted was created as the result of a distributed *Search Request*, and that therefore the *result set* to be deleted is composed of nested sub-*result sets* distributed across different *targets*, the *Delete Result Set Request* must be distributed in the same manner as the original *Search Request* and must therefore be propagated to all the *targets* which hold part of the whole *result set*.

3.5.4.1 Delete Result Set Request

The use of the *delete result set request* by the CIP is presented in detail in 3.5.4.1.

Table 3-16: Delete result set request

ASN.1 Definition	Meaning

`DeleteResultSetRequest ::= SEQUENCE{`
`  referenceId  ReferenceId OPTIONAL,`
`  deleteFunction  [32]  IMPLICIT INTEGER{`
`  list (0),`
`  all (1)},`
`  resultSetList  SEQUENCE OF ResultSetId OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	A **DeleteResultSetRequest** enables an origin to request that the target delete result set(s) created during the current Z-Association. It contains the following information:
	• **referenceId**, which is the reference identifier of the operation initiated by the **DeleteResultSetRequest**.
	• **deleteFunction**, which specifies the kind of deletion requested. The origin can request to delete either a **list** of specified result sets, or **all** the existing result sets created during the current Z-Association.
	• **resultSetList**, which contains the list of result sets created during the current Z-Association which are to be deleted. The **resultSetList** must be provided when the **deleteFunction** specifies that a **list** of results sets is to be deleted.
	• **otherInfo** about the **DeleteResultSetRequest**.

[20] Note that *persistent result sets* are not deleted by the use of the *Result Set Delete facility*, but by the deletion of the *task package* within which the *persistent result set* is defined.

3.5.4.2 Delete Result Set Response

The use of the *delete result set response* by the CIP is presented in detail in 3.5.4.2.

Table 3-17: Delete result set response

ASN.1 Definition	Meaning

`DeleteResultSetResponse ::= SEQUENCE{`
`  referenceId  ReferenceId ~~OPTIONAL~~,`
`  deleteOperationStatus  [0]  IMPLICIT DeleteSetStatus,`
`  deleteListStatuses  [1]  IMPLICIT ListStatuses OPTIONAL,`
`  numberNotDeleted  [34]  IMPLICIT INTEGER OPTIONAL,`
`  bulkStatuses  [35]  IMPLICIT ListStatuses OPTIONAL,`
`  deleteMessage  [36]  IMPLICIT InternationalString OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	A **DeleteResultSetResponse** is a response generated by the target in response to the **DeleteResultSetRequest**. It contains the following information:
	• **referenceId**, which is the reference identifier of the operation initiated by the **DeleteResultSetRequest**.
	• **deleteOperationStatus**, which is the status of the delete request.
	• **deleteListStatuses**, which describes for each result set in the **resultSetList** provided in the **DeleteResultSetRequest** (i.e. when the **deleteFunction** in the request specified that a **list** of result sets was to be deleted) the status of the deletion.
	• **numberNotDeleted**, which indicates the number of results sets which have not been deleted. This is provided only if the **deleteFunction** in the **deleteResultSetRequest** specified that **all** result sets were to be deleted.
	• **bulkStatuses**, which provides a status describing the reason why a result set has not been deleted when the **deleteFunction** in the **deleteResultSetRequest** specified that **all** result sets were to be deleted.
	• **deleteMessage**, which contains a text message regarding the deletion of result sets.
	• **otherInfo** about the **DeleteResultSetResponse**.

`ListStatuses ::= SEQUENCE OF SEQUENCE{`
`  id  ResultSetId,`
`  status  DeleteSetStatus}`
	ListStatuses contains a list of result set **id**s, and specifies for each result set **id** the **status** of the deletion.

ASN.1 Definition	Meaning

`DeleteSetStatus ::= [33] IMPLICIT INTEGER{`
`success (0),`
`resultSetDidNotExist (1),`
`previouslyDeletedByTarget (2),`
`systemProblemAtTarget (3),`
`accessNotAllowed (4),`
`resourceControlAtOrigin (5),`
`resourceControlAtTarget (6),`
`bulkDeleteNotSupported (7),`
`notAllRsltSetsDeletedOnBulkDlte (8),`
`notAllRequestedResultSetsDeleted (9),`
`resultSetInUse (10)}`
	DeleteSetStatus contains the status of the deletion of a result set. The status can be one of the following:
	• **success** indicates that the deletion has been performed successfully.
	• **resultSetDidNotExist** indicates that the result set to be deleted does not exist.
	• **previouslyDeletedByTarget** indicates that the result set to be deleted has already been deleted by the target.
	• **systemProblemAtTarget** indicates that the deletion has not been performed because of some problem at the target.
	• **accessNotAllowed** indicates that the access to the result set to be deleted is not allowed.
	• **resourceControlAtOrigin** indicates that the origin does not have enough resources to perform the deletion.
	• **resourceControlAtTarget** that the target does not have enough resources to perform the deletion.
	• **bulkDeleteNotSupported** indicates that the target does not support the deletion of all result sets.
	• **notAllRsltSetsDeletedOnBulkDlte** indicates that not all result sets have been deleted when the **deleteFunction** in the **DeleteResultSetRequest** was that **all** result sets must be deleted.
	• **notAllRequestedResultSetsDeleted** indicates that not all requested result sets have been deleted when the **deleteFunction** in the **DeleteResultSetRequest** was that a specified **list** of result sets must be deleted.
	• **resultSetInUse** indicates that the result set to be deleted is currently in use.

3.5.5 Access Control Facility

The *Access Control facility* allows a *target* to challenge the identity of an *origin*. This *facility* is used within the CIP to ensure that the identity of the *origin* can be determined by the *target* with an appropriate level of confidence. The CIP requires confidence in the identity of users for certain operations, principally ordering.

The *Access Control facility* consists of a single service, the *Access Control service*. The *Access Control service* in CIP B uses the Z39.50 externally defined format OID *{Z39.50-AccessControl-prompt-1}* for exchanging security challenge and response messages. In addition, the CIP B specification uses the *encrypted* data structure to support the use of shared and public key mechanisms for authenticating *target* or *origin* elements.

3.5.5.1 Access Control Request

The use of the *access control request* by the CIP is presented in detail in Table 3-18.

Table 3-18: Access Control Request

ASN.1 Definition	Meaning

AccessControlRequest ::= SEQUENCE{
referenceId ReferenceId ~~OPTIONAL~~,
securityChallenge CHOICE{
simpleForm [37] IMPLICIT OCTET STRING,
externallyDefined [0] EXTERNAL},
-- use {Z39.50-AccessControl-prompt-1}
otherInfo OtherInformation OPTIONAL}
	AccessControlRequest is sent by the target to the origin as a "challenge" to the identity of the origin. The request may be to identify the origin over a particular operation or over the whole association. The request consists of the following parameters:
	• **referenceID** identifies the association or operation which is being challenged
	• **securityChallenge** parameter containing the security challenge parameters
	• **simpleForm** parameter containing the security challenge
	• **externallyDefined** parameter containing the security challenge in an externally defined format. CIP B uses the OID {Z39.50-AccessControl-prompt-1}
	• **otherInfo** includes additional information to support the challenge. This may be explanatory or part of an externally agreed information exchange.

3.5.5.2 Access Control Response

The use of the *access control response* by the CIP is presented in detail in Table 3-19.

Table 3-19: Access Control Response

ASN.1 Definition	Meaning

AccessControlResponse ::= SEQUENCE{
referenceId ReferenceId ~~OPTIONAL~~,
securityChallengeResponse CHOICE{
simpleForm [38] IMPLICIT OCTET STRING,
externallyDefined [0] EXTERNAL}
OPTIONAL,
-- Optional only in version 3; mandatory
-- in version 2. If omitted (in version 3)
-- then diagnostic must occur.
diagnostic [223] DiagRec OPTIONAL, -- Version 3 only.
otherInfo OtherInformation OPTIONAL}
	AccessControlResponse sent by the origin to the target to a "challenge" to the identity of the origin. The response may identify the origin for a particular operation or over the whole association. The response consists of the following parameters:
	• **referenceID** identifies the association or operation which is being challenged.
	• **securityChallenge** contains the security challenge parameters.
	• **simpleForm** parameter containing the security challenge.
	• **externallyDefined** parameter containing the security challenge in an externally defined format. CIP B uses {Z39.50-AccessControl-prompt-1}.
	• **diagnostic** may contain a diagnostic message.
	• **otherInfo** provides additional information to support the challenge. This may be explanatory or part of an externally agreed information exchange.

3.5.5.3 Access Control Format

The CIP uses the *prompt-1 access control format*. Its use is presented in detail in Table 3-20. The *encrypted* structure is specified by CIP B. This allows the *target* and *origin* to exchange *request* and *response* messages which use shared key or private/public key cryptosytems. The CIP B security approach allows authentication and integrity checking of messages. The approach does not transfer the key or password data over the ICS, either in plain text or encrypted format. The security approach requires that the *target* and *origin* are able to use a common cryptographic algorithm and (if using public key mechanisms) validate the certificates exchanged through a trusted third party Certification Authority.

The CIP B does not support the message exchange between the CIP elements and the Certification Authority, this is outside the protocol and is to be performed using the required exchange mechanism with the Certification Authority.

Table 3-20: Access Control Format

ASN.1 Definition	Meaning

`PromptObject ::= CHOICE{`
`  challenge  [1] IMPLICIT Challenge,`
`  response  [2] IMPLICIT Response}`
	The **PromptObject** is a construction used to contain access control requests or responses. The contents are either:
	• **challenge** structure indicating that the **PromptObject** is a challenge.
	• **response** structure indicating that the **PromptObject** is a response.

`Challenge ::= SEQUENCE OF SEQUENCE {`
`  promptId  [1] PromptId,`
`  -- Target supplies a number (for an enumerated`
`  -- prompt) or string (for a non-enumerated`
`  -- prompt), for each prompt, and the origin`
`  -- returns it in response, for this prompt, so`
`  -- target may correlate the prompt response with`
`  -- the prompt.`
`  defaultResponse  [2] IMPLICIT InternationalString OPTIONAL,`
`  promptInfo  [3] CHOICE{`
`  character  [1] IMPLICIT InternationalString,`
`  encrypted  [2] IMPLICIT Encryption} OPTIONAL,`
`  -- Information corresponding to an enumerated`
`  -- prompt. For example if 'type', within PromptId,`
`  -- is 'copyright', then promptInfo may contain a`
`  -- copyright statement.`
`  regExpr  [4] IMPLICIT InternationalString OPTIONAL,`
`responseRequired  [5] IMPLICIT NULL ~~OPTIONAL~~,`
`allowedValues  [6] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,`
`shouldSave  [7] IMPLICIT NULL OPTIONAL,`
`dataType  [8] IMPLICIT INTEGER{`
`  integer  (1),`
`  date  (2),`
`  float  (3),`
`  alphaNumeric  (4),`
`  url-urn  (5),`
`  boolean  (6)} OPTIONAL,`
`  -- Target telling origin type of data it wants`
`diagnostic  [9] IMPLICIT EXTERNAL OPTIONAL`
	The **Challenge** format contains the parameters passed in a security challenge. the parameters are:
	• **promptId** a number defining the type of challenge (for an enumerated challenge) or a string (for a non-enumerated challenge).
	• **default response** default response expected by the target.
	• **promptInfo** information corresponding to the **promptId**. The content of **promptInfo** depends on the type of the **Challenge**:
	  • If the **Challenge** is for the identification of a user at a remote target, **character** is used. **character** contains a string which indicates to the origin (i.e. the user) that the **userId** at a remote target is requested (rather than the **userId** at the target with which the user has directly established a Z-session) when pass-through authentication (e.g. for pass-through remote ordering) is performed (see also Section 3.5.8.7.2 and 3.9.7.8).
	  • If the **Challenge** is for the authentication of the operation, **encrypted** is used. **encrypted** contains the credentials to be signed by the origin.
	• **regExpr** a regular expression defining the format for the response.
	• **responseRequired** flag indicating whether a response is required.
	• **allowedValues** values that the response is allowed to take.
	• **shouldSave** flag to indicate to the origin that it should save the information requested from the user to respond to the request as it may be requested again.

ASN.1 Definition	Meaning
`}`	• **dataType** enumerated value indicating the type of data expected in response. For the CIP, the expected **dataType** is **alphanumeric**. • **diagnostic** information from the origin to the target describing an error from a previous challenge.

`Response ::= SEQUENCE OF SEQUENCE {`
`promptId [1] PromptId,`
`-- Corresponds to a prompt in the challenge, or`
`-- may be unprompted, for example "newPassword."`
`-- If unprompted, should be "enumerated."`
`-- If this responds to a non-enumerated prompt,`
`-- then nonEnumeratedPrompt should contain the`
`-- prompt string from the challenge.`
`promptResponse [2] CHOICE{`
`string [1] IMPLICIT InternationalString,`
`accept [2] IMPLICIT BOOLEAN,`
`acknowledge [3] IMPLICIT NULL,`
`diagnostic [4] DiagRec,`
`encrypted [5] IMPLICIT Encryption}}
	Response structure containing the information returned by the origin in response to a challenge. The parameters are:
	• **promptId** identifier corresponding to the **promptId** in the challenge, or can be used to identify an unsolicited response, for example to provide new password information
	• **promptResponse** content of the response, one of a choice of data types. For CIP B the data type depends on the type of the **Challenge**:
	  • If the **Challenge** was for the identification of a user at a remote target, **string** is used. **string** contains a string which indicates to the target the user's **userId** at the remote target which sent the **Challenge** (rather than the **userId** at the target with which the user has directly established a Z-session) when pass-through authentication (e.g. for pass-through remote ordering) is performed (see also Section 3.5.8.7.2 and 3.9.7.8).
	  • If the **Challenge** was for the authentication of the operation, **encrypted** is used. **encrypted** contains the credentials from the target encrypted by the origin.

`PromptId ::= CHOICE{`
`enummeratedPrompt [1] IMPLICIT SEQUENCE{`
`type [1] IMPLICIT INTEGER{`
`groupId (0),`
`userId (1),`
`password (2),`
`newPassword (3),`
`copyright (4),`
`-- When type on Challenge is 'copyright',`
`-- promptInfo has text of copyright message`
`-- to be displayed verbatim to the user.`
`-- If promptResponse indicates 'acceptance',`
`-- this indicates the user has been shown,`
`-- and accepted, the terms of the copyright.`
`-- This is not intended to be legally binding,`
`-- but provides a good-faith attempt on the`
`-- part of the target to inform the user of`
`-- the copyright.`
`sessionId (5)},`
`suggestedString [2] IMPLICIT`
`InternationalString OPTIONAL},`
`nonEnumeratedPrompt [2] IMPLICIT InternationalString}`
	Format for the **PromptId** describing what type of challenge or response has been created. The **PromptId** is either:
	• an **enummeratedPrompt** containing
	  • the **type** of the **PromptId**, i.e. a **groupId**, a **userId**, a **password**, a **newPassword** or a **copyright**.
	  • a **suggestedString** to be provided in the prompt to the user. This **suggestedString** is used, for example, to clearly indicate to the user that his **userId** at a remote target is requested (rather than his userId at the target with which he has directly established a Z-session) when pass-through authentication (e.g. for pass-through remote ordering) is performed (see also Section 3.5.8.7.2 and 3.9.7.8).
	• a **nonEnumeratedPrompt**, which is a string containing the prompt.

ASN.1 Definition	Meaning

`Encryption ::= SEQUENCE{`
`  cryptType  [1] IMPLICIT OCTET STRING OPTIONAL,`
`  credential  [2] IMPLICIT OCTET STRING OPTIONAL,`
`  --random number, SALT, or other factor`
`  data  [3] IMPLICIT OCTET STRING}`
	Encryption specifies the format of encrypted data exchanges. The format provides three parameters:
	• **cryptType** identifies the type of algorithm used to encrypt the data. For CIP B shared key security the algorithm to be used is RFC 1321 (MAC-MD5). Other algorithms for public key cryptography are by arrangement between entities, they may include DES, SHA and RSA algorithms. The size of the key used must also be specified where appropriate but must in any case be understood by both parties. In addition, the Certificate Authority must be specified for public key authentication.
	• **credential** the data being signed. For CIP B shared key security the credential is supplied by the challenging entity (the target) and is composed of:
	  • timestamp;
	  • userId;
	  • Operation.
	  Where the Operation field is the operation that triggered the challenge.
	• **data** for **AccessControlResponse** messages, this field contains the digital signature. For **AccessControlRequest** messages, this field must be NULL.
	The exact details on **Encryption** (e.g. credential details) are described in Section 3.9.7.

3.5.6 Accounting/Resource Control Facility

The *Accounting and Resource Control facility* provides services to allow the *origin* and *target* to exchange *requests* on the usage and status of *resources* at the *target*. The *facility* provides information to the *origin* concerning the progress of queries.

The *Accounting and Resource Control facility* consists of three services: the R*esource Control service*, the *TriggerResource Control service* and the R*esource-report service*. These *services* are described in the following subsections.

3.5.6.1 Resource Control Service

The *resource control request* (presented in Table 3-21) is sent by the *target* to the *origin* when *resource control* is in effect. *Resource control* is agreed between the *origin* and *target* at the start of an *Association*. The *request* allows the *target* to provide information concerning the status of an *operation* and also to indicate whether a *response* is required from the *origin*. When a *resource control request* is performed for an operation which was forwarded down the collection hierarchy (such as a *Search* or *Present request*), the *resource control request* reflects the hierarchical structure of the operation. For the node in the collection hierarchy for which the *resource control request* is sent, the *resource control request* contains the report for the current node and also contains the consolidated report for all the children of the node in the collection hierarchy.

A *resource-control response* (presented in

Table 3-22)allows the *origin* to indicate whether to continue an *operation* or not. It can also indicate whether a *result set* is wanted by the *origin*.

Table 3-21: Resource Control Request

ASN.1 Definition	Meaning

`ResourceControlRequest ::= SEQUENCE{`
`  referenceId  ReferenceId ~~OPTIONAL~~,`
`  suspendedFlag  [39]  IMPLICIT BOOLEAN OPTIONAL,`
`  resourceReport  [40]  ResourceReport ~~OPTIONAL~~,`
`  partialResultsAvailable  [41]  IMPLICIT INTEGER{`
`  subset (1),`
`  interim (2),`
`  none (3)} OPTIONAL,`
`  responseRequired  [42]  IMPLICIT BOOLEAN,`
`  triggeredRequestFlag  [43]  IMPLICIT BOOLEAN OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	The **ResourceControlRequest** structure consists of the following parameters:
	• **referenceId** the reference identifier of the operation.
	• **suspendedFlag** a flag to denote whether the target has suspended the operation pending a response from the origin.
	• **resourceReport** conveys information about the operation. When the **ResourceControlRequest** is performed for an operation which was forwarded down the collection hierarchy, **resourceReport** contains the report for the current node in the collection hierarchy.
	• **partialResultsAvailable** indicates whether the origin can request partial results of a search from the target.
	• **responseRequired** indicates whether the origin needs to respond to this request.
	• **triggeredRequestFlag** this is an optional parameter to indicate whether the request has been created in response to a trigger-resource-control request.
	• **otherInfo** provides other information about the children of the current collection node when the **ResourceControlRequest** is performed for an operation which was forwarded down the collection hierarchy. This is performed using the **CIPSpecificInfo** EXTERNAL and selecting **childrenResourceReport**, as defined in Appendix E.6.1.

ASN.1 Definition	Meaning
`ResourceReport ::= EXTERNAL`	The **ResourceReport** structure contains the report about the resources. The **ResourceReport** format is externally defined. The selection of the external definition is based on the type of operation for which the **ResourceReport** is established: if the operation is a Search operation, {Z39.50-UserFormat-searchResult-1}is selected. Otherwise, {Z39.50-ResourceReport-resource-1} is selected. Moreover, if the **ResourceControlRequest** was triggered by a **TriggerResourceControlRequest**, the preferred resource format defined in **prefResourceReportFormat** may be considered. The resource report formats are presented in Section 3.5.6.4.

Table 3-22: Resource Control Response

ASN.1 Definition	Meaning

`ResourceControlResponse ::= SEQUENCE{`
`referenceId ReferenceId OPTIONAL,`
`continueFlag [44] IMPLICIT BOOLEAN,`
`resultSetWanted [45] IMPLICIT BOOLEAN OPTIONAL,`
`otherInfo OtherInformation OPTIONAL}`
	The **ResourceControlResponse** is provided by the *target* in response to a **resourceControlRequest** from the origin. The parameters are:
	• **referenceId** is the identifier for the operation.
	• **continueFlag** is used to denote whether the origin wants the target to continue with the operation.
	• **resultSetWanted** indicates whether a result set is wanted by the origin..
	• **otherInfo** provides other information.

3.5.6.2 Trigger Resource Control Service

The *Trigger Resource Control service* is used by the *origin* to request a *status report* or change the status of an existing operation. The *origin* may request a *status report* (which will result in a *resource control request* with no acknowledgement required) or it may request full *resource control* (which will require the *origin* to provide full *resource control* on an *operation*. Finally the *Trigger Resource Control service* may be used to cancel an operation started by the *origin*. When a *Trigger Resource Control request* is performed for an operation which was forwarded down the collection hierarchy (such as a *Search* or *Present request*), the *Trigger Resource Control request* is forwarded in the same manner as this operation down the collection hierarchy so that the result of the request (i.e. the *status report* provided by the *resource control request*) reflects the hierarchical structure of the operation.

Table 3-23: Trigger Resource Control Request

ASN.1 Definition	Meaning

`TriggerResourceControlRequest ::= SEQUENCE{`
`  referenceId  ReferenceId OPTIONAL,`
`  requestedAction  [46]  IMPLICIT INTEGER{`
`  resourceReport (1),`
`  resourceControl (2),`
`  cancel (3)},`
`  prefResourceReportFormat  [47]  IMPLICIT ResourceReportId OPTIONAL,`
`  resultSetWanted  [48]  IMPLICIT BOOLEAN OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	The **TriggerResourceControlRequest** is provided by the origin to the target to request that the target change the status of the reporting on an operation. This is a non confirmed service. The parameters are:
	• **referenceId** is the reference id is of the operation.
	• **requestedAction** contains the requested action to be taken, either for a report, to turn on full resource control or to cancel the operation.
	• **prefResourceReportFormat** specifies the preferred format for the report.
	• **resultSetWanted** flag to indicate whether a result set is required to be stored when the operation is search and the trigger-resource-control is requesting that the operation be cancelled. This will result in a partial result set being stored.
	• **otherInfo** provides other information..

3.5.6.3 Resource Report Service

A *resource report* describes the status of an *operation* that has been completed by the *target* on behalf of the *origin*. The origin may request a *resource report* from the *target* and can specify the format that it wishes to receive the report in. The *resource report request* is presented in Table 3-24, and the corresponding *resource control response* is presented in Table 3-25).

Table 3-24: Resource Report Request

ASN.1 Definition	Meaning

`ResourceReportRequest ::= SEQUENCE{`
`  referenceId  ReferenceId OPTIONAL,`
`  opId  [210]  IMPLICIT ReferenceId OPTIONAL,`
`  prefResourceReportFormat  [49]  IMPLICIT ResourceReportId OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	The **ResourceReportRequest** allows an origin to request a target to produce a resource report. The parameters are:
	• **referenceId** the reference id of the operation.
	• **opId** is the reference id of the operation being reported.
	• **preResourceReportFormat** includes the resource report format requested.
	• **otherInfo** provides other information.

Table 3-25: Resource Report Response

ASN.1 Definition	Meaning

`ResourceReportResponse ::= SEQUENCE{`
`  referenceId  ReferenceId ~~OPTIONAL~~,`
`  resourceReportStatus  [50]  IMPLICIT INTEGER{`
`  success  (0),`
`  partial  (1),`
`  failure-1  (2),`
`  failure-2  (3),`
`  failure-3  (4),`
`  failure-4  (5),`
`  failure-5  (6),`
`  failure-6  (7)},`
`  resourceReport  [51]  ResourceReport OPTIONAL,`
`  otherInfo  OtherInformation OPTIONAL}`
	The **ResourceReportResponse** is a response generated by the target in response to the **ResourceReportRequest**. The parameters are:
	• **referenceId** is the reference identifier of the operation.
	• **resourceReportStatus** is a flag to denote the status of the report. The values are: **success**; **partial**; **failure-1** (target unable to provide resource report); **failure-2** (operation terminated due to resource constraints); **failure-3** (access control failure); **failure-4** (unspecified failure); **failure-5** (no know operation with this id); **failure-6** (active operation with the passed id).
	• **resourceReport** is used to convey the actual report.
	• **otherInfo** provides other information.

3.5.6.4 Resource Report Format

The CIP uses the *resource-1 report format*, presented in Table 3-26, and the searchResult-1 user information format, presented in Table 3-27, for the *resource reports*.

Table 3-26: resource-1 Resource Report Format

ASN.1 Definition	Meaning

`ResourceReport ::= SEQUENCE{`
`estimates  [1] IMPLICIT SEQUENCE OF Estimate,`
`message  [2] IMPLICIT InternationalString}`
	ResourceReport contains information about the status of an operation. Either a simple message or an estimates structure is provided.

`Estimate ::= SEQUENCE{`
`type  [1] IMPLICIT EstimateType,`
`value  [2] IMPLICIT INTEGER,  -- the actual estimate`
`currency-code  [3] IMPLICIT INTEGER OPTIONAL`
`-- code for representation of currencies defined in ISO`
`-- 4217-1990.  Applicable only to monetary estimates.`
`}`
	Estimate provides information to the origin about the status of an operation.. The parameters are:
	• **type** contains the type of report being provided
	• **value** contains the actual value of the estimate
	• **currency-code** is the currency code for financial estimates.

ASN.1 Definition	Meaning

`EstimateType ::= INTEGER{`
`currentSearchRecords (1),`
`-- estimated no. records in current (incomplete) result set for search`
`finalSearchRecords (2),`
`-- estimated no. records that will be in result set if search completes`
`currentPresentRecords (3),`
`-- estimated number of records in current (incomplete) set of records to be`
`-- returned on Present`
`finalPresentRecords (4),`
`-- estimated number of records that will be in the set of records to be`
`-- returned by Present if Present completes`
`currentOpTimeProcessing (5),`
`-- processing time (in .001 CPU seconds) used by operation so far`
`finalOpTimeProcessing (6),`
`-- estimated total processing time (in .001 CPU seconds) that will be used`
`-- by this operation if it completes`
`currentAssocTime (7),`
`-- estimated processing time used by association (in .001 CPU sec.)`
`currentOperationCost (8),`
`-- estimated cost for this operation so far`
`finalOperationCost (9),`
`-- estimated cost for this operation if it completes`
`currentAssocCost (10),`
`-- estimated cost for this association so far`
`finalOpTimeElapsed (11),`
`-- estimated elapsed time for operation if it completes (in .001 sec.)`
`percentComplete (12),`
`-- estimated percent complete`
`currentSearchAssocCost (13),`
`-- estimated search cost for association so far`
`currentPresentAssocCost (14),`
`-- estimated present cost for this association so far`
`currentConnectAssocCost (15),`
`-- estimated connect time cost for association so far`
`currentOtherAssocCost (16)`
`-- estimated other cost (not included in 13-15) for association so far`
`}`
	EstimateType indicates the type of estimate being provided. The parameter meanings are defined in the comments associated with the ASN.1 structure. Note that the cost options are not used in CIP-B. The elements of **EstimateType** are:
	• **currentSearchRecords** indicating the estimated number of records in the current (incomplete) result set.
	• **searchfinalSearchRecords** indicating the estimated number of records that will be in the result set if the search completes.
	• **currentPresentRecords** indicating the estimated number of records in the current (incomplete) set of records to be returned on Present.
	• **finalPresentRecords** indicating the estimated number of records that will be in the set of records to be returned by Present if Present completes
	• **currentOpTimeProcessing** indicating the processing time (in .001 CPU seconds) used by operation so far.
	• **finalOpTimeProcessing** indicating the estimated total processing time (in .001 CPU seconds) that will be used by this operation if it completes.
	• **currentAssocTime** indicating the estimated processing time used by association (in .001 CPU sec.).
	• **currentOperationCost** indicating the estimated cost for this operation so far.
	• **finalOperationCost** indicating the estimated cost for this operation if it completes.
	• **currentAssocCost** indicating the estimated cost for this association so far.
	• **finalOpTimeElapsed** indicating the estimated elapsed time for the operation if it completes (in .001 sec.).
	• **percentComplete** indicating the percentage of completion.
	• **currentSearchAssocCost** indicating the estimated search cost for the association so far.
	• **currentPresentAssocCost** indication the estimated present cost for this association so far.
	• **currentConnectAssocCost** indicating the estimated connect time cost for association so far.
	• **currentOtherAssocCost** indicating the estimated other cost (not included in 13-15) for association so far.

Table 3-27: searchResult-1 User Information Format

ASN.1 Definition	Meaning

`SearchInfoReport ::= SEQUENCE OF SEQUENCE{`
`  subqueryId [1] IMPLICIT InternationalString OPTIONAL,`
`  -- shorthand identifier of subquery`
`  fullQuery [2] IMPLICIT BOOLEAN,`
`  -- 'true' means this is the full query;`
`  -- 'false', a sub-query`
`  subqueryExpression [3] QueryExpression OPTIONAL,`
`  -- A subquery of the query as submitted.`
`  -- May be whole query; if so, "fullQuery"`
`  -- should be 'true'.`
`  subqueryInterpretation [4] QueryExpression OPTIONAL,`
`  -- how target interpreted subquery`
`  subqueryRecommendation [5] QueryExpression OPTIONAL,`
`  -- target-recommended alternative`
`  subqueryCount [6] IMPLICIT INTEGER OPTIONAL,`
`  -- Number of records for this subQuery,`
`  -- across all of the specified databases.`
`  -- (If during search, via resource`
`  -- control,number of records so far).`
`  subqueryWeight [7] IMPLICIT IntUnit OPTIONAL,`
`  -- relative weight of this subquery`
`  resultsByDB [8] IMPLICIT ResultsByDB OPTIONAL}`
	SearchInfoReport contains information about the status of a search operation. The parameters are:
	• **subqueryId** identifying the query. This must be the identifier of the collection targeted by the search.
	• **fullQuery** indicating whether the report is for the full query or a sub-query. This must be set to 'true'.
	• **subqueryExpression** indicating the query expression. This must be the full query.
	• **subQueryInterpretation** indicating how the target interpreted the query.
	• **subQueryRecommendation** indicating an alternative for the query.
	• **subQueryCount** indicating the number of records matching the query (so far, if the query is not completed).
	• **subQueryWeight** indicating the relative weight of the query.
	• **resultsByDB** indicating, for each collection, the number of records.

`ResultsByDB ::= SEQUENCE OF SEQUENCE{`
`  databases [1] CHOICE{`
`  all [1] IMPLICIT NULL,`
`  -- applies across all of the databases`
`  -- in Search PDU`
`  list [2] IMPLICIT SEQUENCE OF DatabaseName`
`  -- applies across all databases in`
`  -- this list`
`  },`
`  count [2] IMPLICIT INTEGER OPTIONAL,`
`  -- Number of records for query component (and, as`
`  -- above, if during search, via resource control,`
`  -- number of records so far).`
`  resultSetName [3] IMPLICIT InternationalString OPTIONAL`
`  -- Target-assigned result set by which subQuery is`
`  -- available. Should not be provided unless`
`  -- processing for this query component is concluded`
`  -- (i.e., when this report comes during search, via`
`  -- resource control, as opposed to after search,`
`  -- via additionalSearchInfo).`
`  }`
	ResultsByDB contains information about the results on a per collection basis. The parameters are:
	• **databases** indicating whether the results apply for **all** the collections searched or for an explicit **list** of collections.
	• **count** indicating the number of records matching the query (so far, if the query is not completed).
	• **resultSetName** indicating the name of the result set in which the results are available.

`QueryExpression ::= CHOICE {`
`  term [1] IMPLICIT SEQUENCE{`
`  queryTerm [1] Term,`
`  termComment [2] IMPLICIT InternationalString OPTIONAL},`
`  query [2] Query}`
	QueryExpression contains information about the query. The parameters are:
	• **term** describing the term used in the query.
	• **query** containing the query which was performed.

3.5.7 Extended Services Facility

The Z39.50 Extended Services used by the CIP, and their usage by the CIP, is presented in the following sub-sections as follows:

- Section 3.5.7.1 introduces the concept of the *Extended Services*.
- Section 3.5.7.2 presents the *Persistent Result Set Extended Service*, which allows *origins* to store result sets persistently across Z-Associations.
- Section 3.5.7.3 presents the *Persistent Query Extended Service*, which allows *origins* to store query definitions persistently across Z-Associations.
- Section 3.5.7.4 presents the *Periodic Query Schedule Extended Service*, which allows *origins* to define persistent queries that are performed periodically.
- Section 3.5.7.5 presents the *Database Update Extended Service*, which allows *origins* to update the content of a database accessible via Z39.50.

Additionally:

- Section 3.5.8 presents the CIP Order *Extended Service*, which allows *origin* to process orders via the CIP[21].

3.5.7.1 Introduction

3.5.7.1.1 Overview

The *Extended Services facility* enables an *origin* to perform a *service* which is not defined by Z39.50, nor is executed within its scope, at a *target*. The *Extended Services facility* therefore provides a mechanism to define and monitor tasks that are executed outside Z39.50.

An *Extended Service* (ES) defines a particular *task* which is related to information retrieval but is not defined as a *service* within Z39.50. It allows an *origin* to create, modify or delete *task packages*, which are maintained by the *target* in a special database, the *Extended Services database*.

The *task* defined in a *task package* (and performed externally) depends on the particular *Extended Service* used. However, the handling of a *task package* is identical for all *Extended Services* and is performed as follows:

- The *origin* sends an *ES request* to the *target*, which requests the execution of a *task*. The *ES request* includes all the service-dependent parameters necessary for the *target* to construct the *task package*.
- The *target* checks the request's validity and the user's access rights to perform the *task* and sends an *ES response*, which indicates either the acceptance of the requested *task*, or the rejection of the *task*.

The execution of the *ES operation* described above results in the creation of a *task package*, which is represented by a *database record* in the *ES database*. Note however that, whilst an *ES request* may result in the initiation of a *task*, the *task* itself is not considered as part of the *operation*. The *ES response* therefore does not necessarily signal the completion of the *task*, which may have a lifetime which exceeds the *Z-Association* during which it is initiated.

[21] A custom CIP Order *Extended Service* has been defined instead of using the Z39.50 *Item Order ES* because the latter could not easily be adapted to fulfil CIP order requirements.

Also, as for any *Z39.50 operation*, an *origin* may receive the *ES response* corresponding to an *ES request* only during the *Z-Association* during which the *ES request* is issued. However, the abortion of an *ES operation* (i.e. the closure of the *Z-Association* in which the *ES request* is performed before the reception of its corresponding *ES response*) has no effect on the disposition or the processing of the *task* initiated by an *ES request*. In other words, even if an *ES operation* is aborted, the *task* spanned by the *ES request* itself is not aborted, but is performed normally. However, as in this case no *ES response* can be sent to the *origin*, the *origin* must search the *ES database* in order to retrieve the information that would otherwise have been returned by the *ES response*.

A *task package* contains parameters. Some parameters, typically used for the management and monitoring of the *task package*, are common to all *Extended Services*. Other parameters, typically used for the definition of the *task* to be executed, are specific to a particular *Extended Service*. This is illustrated with an OMT functional model in Figure 3-8.

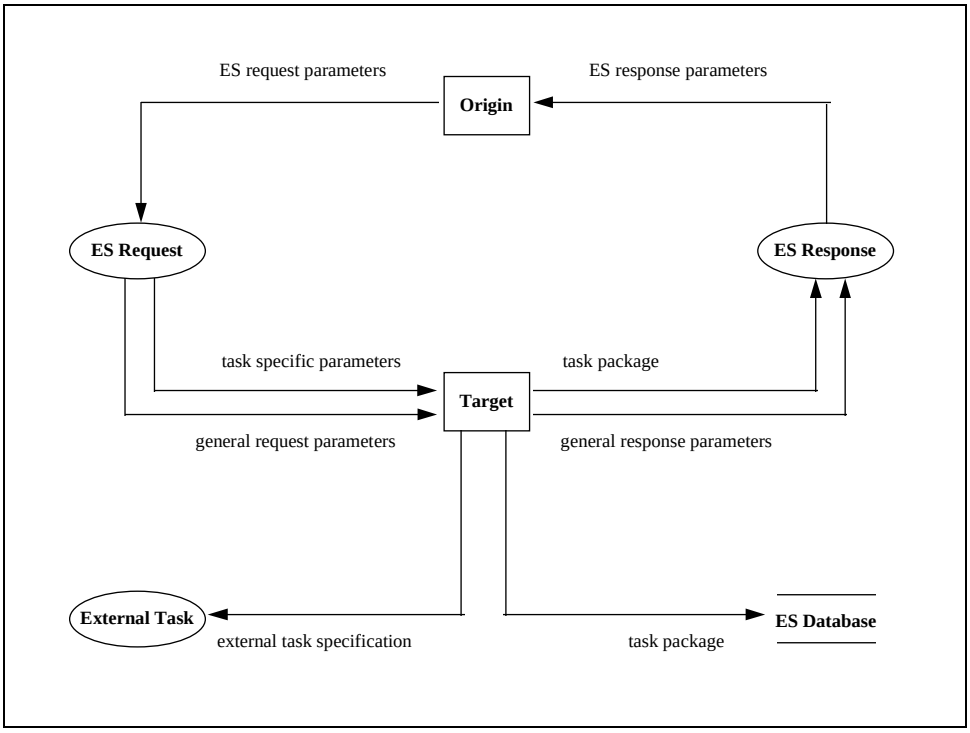

Figure 3-8: Extended Services Functional Model

Some parameters are supplied by the *origin* as parameters in the *ES request* (the *task specific parameters*) and are used by the *target* as input for the *external task*. The *target* keeps some of the parameters provided by the *origin* unchanged (from the *origin part to keep*). It may also override some parameters provided by the *origin* (from the *origin part not to keep*) and supply additional parameters (in the *target part*). The *target* uses the parameters supplied by the *origin* as well as any additional parameters generated by the *target* itself to form the *task package*, which is then returned by the *target* to the *origin* in the *ES response*. This is illustrated in Figure 3-9.

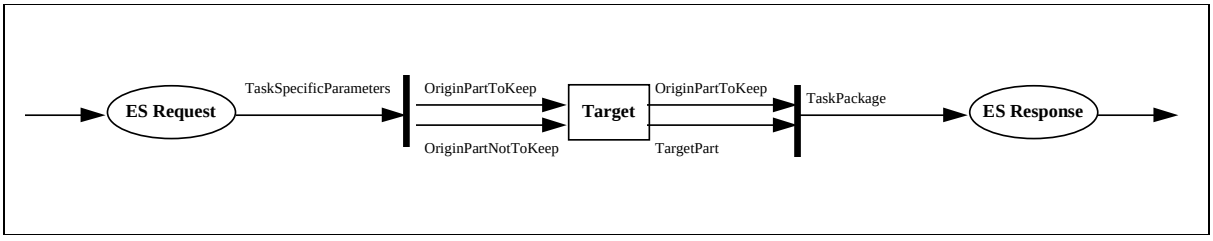

Figure 3-9: Extended Services Task Specific Parameters

A *task package* belongs to the user who created it, who is known as the owner of the *task package*. Moreover, *permissions* are assigned to a *task package*. Full access and use of the *task package* is granted to its *owner*. Additionally, the *owner* can manage the *permissions* and provide other users, or groups or users, access to the *task package*. The following *permissions* may be assigned to a user in relation to the access and use of a *task package*:

- ***Delete***: deletion of a *task package*.
- ***Modify content***: modification of the content of a *task package*.
- ***Modify Permissions***: modification of the permissions for the access and use of a *task package*.
- ***Present***: use of a *task package* with the *Present service*.
- ***Invoke***: invocation of a *task package* via another *Extended Service*.

3.5.7.1.2 Extended Services Database

Targets which support the *Extended Services facility* must provide access to the *Extended Services database* (*ES database*) *IR-Extend-1*. The records in the *ES database* are the *task packages*, which are constructed from the information provided by the *ES request* and completed with the information resulting from the execution of the *task*.

The *ES database* appears to an *origin* as any other database supported by a *target* and may be accessed using the *Search* and *Retrieval facilities*. The search of the *ES database* is specified in [Z3950], Appendix 3 ATR. The structure of the records (i.e. the *task package* structure) is defined in the various *Extended Services* specifications. The retrieval of the *records* is performed according to the *Full*, *Identification*, *UniqueName*, *Permissions* and *Status* element sets defined in [Z3950], Section 3.2.9.2 and the *ES record syntax* presented in Section 3.5.7.1.5.

In the CIP, the *target* shall create a *task package* immediately upon reception of an *ES request*. Moreover, the *target* shall retain a *task package* at least until the requested *task* has completed. Additionally, for some *Extended Services*, the *target* shall retain the *task package* until the *origin* explicitly requires that it be deleted.

3.5.7.1.3 Extended Service Request

The *Extended Services request* is structured as follows. A semantic definition of each required and optional part is provided adjacent to the appropriate formal ASN.1 definition.

Table 3-28: Extended Services Request

ASN.1 Definition	Meaning

`ExtendedServicesRequest ::= SEQUENCE{`
`referenceId ReferenceId ~~OPTIONAL~~,`
`function [3] IMPLICIT INTEGER {`
`create (1),`
`delete (2),`
`modify (3)},`
`packageType [4] IMPLICIT OBJECT IDENTIFIER,`
`packageName [5] IMPLICIT InternationalString OPTIONAL,`
`-- PackageName mandatory for 'modify' or`
`-- 'delete'; optional for 'create'.`
`-- Following four parameters mandatory`
`-- for 'create'; should be included on`
`-- 'modify' if being modified; not needed`
`-- on 'delete'.`
`userId [6] IMPLICIT InternationalString ~~OPTIONAL~~,`
`retentionTime [7] IMPLICIT IntUnit OPTIONAL,`
`permissions [8] IMPLICIT Permissions OPTIONAL,`
`description [9] IMPLICIT InternationalString OPTIONAL,`
`taskSpecificParameters [10] IMPLICIT EXTERNAL OPTIONAL,`
`-- Mandatory for 'create'; included on`
`-- 'modify' if specific parameters being`
`-- modified; not necessary on 'delete'.`
`-- For the 'EXTERNAL,' use OID of`
`-- specific ES definition and select`
`-- CHOICE [1]: 'esRequest'.`
`waitAction [11] IMPLICIT INTEGER{`
`wait (1),`
`waitIfPossible (2),`
`dontWait (3),`
`dontReturnPackage (4)},`
`elements ElementSetName OPTIONAL,`
`otherInfo OtherInformation OPTIONAL}`
	An **ExtendedServicesRequest** allows an origin to perform tasks which are executed outside the scope of Z39.50. The specification of an ES request contains the following elements:
	• The **referenceId** is the reference identifier assigned to the ES operation.
	• The **function** defines the operation that is performed on the task package. The following three functions are supported:
	  • **create** a new task package.
	  • **modify** the content of an existing task package and/or the permissions for its access. Note that only the parameters explicitly provided are modified.
	  • **delete** a task package.
	• The **packageType** determines the kind of ES that must be executed.
	• The **packageName** is the name of the task package.
	• The **userId** is the user identifier of the owner of the task package. The **userId** is the same as used during Init request. **userId** could be 'anonymous'. The **userId** is used to state who owns the task package. If the origin returns a **userId**, which is not the same as during Init the target sends a diagnostic message. A "guest" user can order if allowed by the agency
	• The **retentionTime** is the duration during which the task package must be retained by the target.
	• The **permissions** specify the users, or groups of users, who may access the task package and the access rights granted to them. By default, only the owner of a task package is granted full access to the task package.
	• The **description** describes the content of the task package.
	• The **taskSpecificParameters** contains all the parameters that are specific to the Extended Service that must be executed. See 'esRequest' in Table 3-31.
	• The **waitAction** indicates if the task package must be included in the ES Response by the target. The following actions are supported by the CIP:
	  • **wait** indicates that the target must wait for the task to be performed before issuing an ES Response, which shall contain the task package.
	  • **waitIfPossible** indicates that the target must perform the task before sending the ES Response and include the task package in the response if possible, but that otherwise it may return the ES Response earlier and without the task package.
	  • **dontWait** indicates that the target does not have to wait for the task to be performed before issuing an ES Response, which will therefore not

ASN.1 Definition	Meaning
	necessarily contain the task package. • **dontReturnPackage**: indicates that the target must not include the task package in the ES Response, whatever the timing of the execution of the task. • The **elements** specifies the element set name for the task package as specified in Section 3.5.7.1.1. • The **otherInfo** is used for the specification of additional information.

`Permissions ::= SEQUENCE OF SEQUENCE{`
`  userId  [1] IMPLICIT InternationalString,`
`  allowableFunctions  [2] IMPLICIT SEQUENCE OF INTEGER{`
`  delete  (1),`
`  modifyContents  (2),`
`  modifyPermissions  (3),`
`  present  (4),`
`  invoke  (5)}}`
	The **Permissions** specifies the list of functions that a user, or a group of users, identified by the **userId** is allowed to perform on a task package.
	The following access rights may be granted:
	• **delete** grants the right to delete the task package.
	• **modifyContents** grants the right to modify the content of the task package.
	• **modifyPermissions** grants the right to modify the permissions to access the task package.
	• **present** grants the right to retrieve the content of the task package (via the Present Service)[22].
	• **invoke** grants the right to invoke the task package within another task package.

[22] This can be used, for instance, to grant the right to another user to access a persistent unregistered collection, i.e. a *persistent result set* or *persistent query*.

3.5.7.1.4 Extended Service Response

The *Extended Services response* is structured as follows. A semantic definition of each required and optional part is provided adjacent to the appropriate formal ASN.1 definition.

Table 3-29: Extended Services Response

ASN.1 Definition	Meaning

`ExtendedServicesResponse ::= SEQUENCE{`
`  referenceId  ReferenceId ~~OPTIONAL~~,`
`  operationStatus  [3]  IMPLICIT INTEGER{`
`  done (1),`
`  accepted (2),`
`  failure (3)},`
`  diagnostics  [4]  IMPLICIT SEQUENCE OF DiagRec OPTIONAL,`
`  taskPackage  [5]  IMPLICIT EXTERNAL OPTIONAL,`
`  -- Use OID: {Z39-50-recordSyntax (106)}`
`  -- and corresponding syntax. For the`
`  -- EXTERNAL, 'taskSpecific,' within that`
`  -- definition, use OID of the specific`
`  -- es, and choose [2], 'taskPackage'.`
`  otherInfo  OtherInformation OPTIONAL}`
	An **ExtendedServicesResponse** allows a target to send back the response related to the execution of tasks executed outside the scope of Z39.50 and initiated by an ES Request. The ES Response contains the following elements:
	• The **referenceId** is the reference identifier assigned to the ES operation.
	• The **operationStatus** is the status of the ES operation. The ES operation can be:
	  • **done**, i.e. the task is complete and the results are included in the task package.
	  • **accepted**, i.e. the request was accepted and the task is queued for processing or is being processed.
	  • **failure**, i.e. the request was refused. In this case, diagnostic(s) must be provided to explain the reason of the refusal.
	• **diagnostics** are provided if the ES Request is refused.
	• The **taskPackage** includes the actual task package and is provided if the ES operation is **done**. The portion of the task package that is returned depends on the **element** parameter provided in the ES Request. See 'elements' in Table 3-28.
	• The **otherInfo** is used for the return of additional information regarding the ES operation.

3.5.7.1.5 Extended Service Record Syntax

The *Extended Services record syntax* is structured as follows. A semantic definition of each required and optional part is provided adjacent to the appropriate formal ASN.1 definition.

Table 3-30: Record Syntax for Extended Services Task Package

ASN.1 Definition	Meaning

TaskPackage ::= SEQUENCE{
packageType [1] IMPLICIT OBJECT IDENTIFIER,
-- oid of specific ES definition
packageName [2] IMPLICIT InternationalString OPTIONAL,
userId [3] IMPLICIT InternationalString ~~OPTIONAL~~,
retentionTime [4] IMPLICIT IntUnit OPTIONAL,
permissions [5] IMPLICIT Permissions OPTIONAL,
description [6] IMPLICIT InternationalString OPTIONAL,
targetReference [7] IMPLICIT OCTET STRING OPTIONAL,
creationDateTime [8] IMPLICIT GeneralizedTime OPTIONAL,
taskStatus [9] IMPLICIT INTEGER{
pending (0),
active (1),
complete (2),
aborted (3)},
packageDiagnostics [10] IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
taskSpecificParameters [11] IMPLICIT EXTERNAL
-- Use oid for specific ES definition
-- (same oid as packageType above)
-- and select [2] "taskPackage."
}
	A **TaskPackage** consists in the following elements:
	• The **packageType** , which identifies the type of the task package.
	• The **packageName**, which is the name of the task package supplied by the origin.
	• The **userId**, which is the name of the owner of the task package.
	• The **retentionTime**, which is the duration the target must keep the task package.
	• The **permissions**, which lists the user, or groups of users, who are granted access to the task package together which the access rights granted to them.
	• The **description**, which provides a description of the content of the task package.
	• The **targetReference**, which is the task package identifier supplied by the target.
	• The **creationDateTime**, which is the date and time when the task package was created.
	• The **taskStatus**, which is the status of the task. The value of the status is "pending", "active", "complete" or "aborted".
	• The **packageDiagnostics**, which are diagnostics related to the task package.
	• The **taskSpecificParameters**, which are the task specific parameters. See 'taskPackage' in Table 3-31.

3.5.7.1.6 Extended Services Mechanism

The *task specific parameters* used by each specific *Extended Service* are all provided according to the following mechanism:

- An *ES Request* includes the input *task specific parameters* by selecting the appropriate *Extended Service* definition via its *OID* and, within the *Extended Service*, by choosing the *ES request parameters*.
- An *ES Response*, or a *Present Response* to an *ES Database* search, includes the output *task specific parameters* by selecting the appropriate *Extended Service* definition via its *OID* and, within the *Extended Service*, by choosing the *task package parameters*.

This mechanism is implemented using the template formally described in Table 3-31.

Table 3-31: Extended Services Mechanism

ASN.1 Definition	Meaning

`ExtendedService ::= CHOICE{`
`  esRequest  [1] IMPLICIT SEQUENCE{`
`  toKeep  [1] OriginPartToKeep OPTIONAL,`
`  notToKeep  [2] OriginPartNotToKeep OPTIONAL},`
`  taskPackage  [2] IMPLICIT SEQUENCE{`
`  originPart  [1] OriginPartToKeep OPTIONAL,`
`  targetPart  [2] TargetPart OPTIONAL}}`
	Each specific **ExtendedService** contains the following definition for the parameters of a task package:
	• **esRequest** consists of all the service specific input task parameters supplied by the origin in the ES Request (see **taskSpecificParameters** in Table 3-30). Two kinds of input task parameters are distinguished:
	  • **toKeep** contains the parameters that are to be retained in the task package.
	  • **notToKeep** contains the parameters that are not to be retained in the task package or that may be overridden by the target (i.e. for parameters which are both in **notToKeep** and **targetPart**).
	• **taskPackage** consists of all the service specific output task parameters that are contained in a task package (see **taskPackage** in Table 3-29 and **taskSpecificParameters** in Table 3-30). Two kinds of output task parameters are distinguished:
	  • **originPart** are the **toKeep** parameters supplied by the origin in **esRequest** in the ES Request.
	  • **targetPart** are the parameters supplied by the target.

3.5.7.2 Persistent Result Set Extended Service

The *Persistent Result Set Extended Service* allows an *origin* to request that the *target* create a *persistent result set* (i.e. a *result set* which lifetime may span *Z-Associations*) from a *result set* resulting from a local search[23] performed during the current *Z-Association*. *Persistent result sets* are also referred to as persistent unregistered collections.

The *Persistent Result Set ES*, presented in Table 3-32, is used by the CIP to create from a temporary *result set* a static persistent unregistered collection (i.e. a collection which contents does not change over time). Note that such a persistent unregistered collection is merely a *result set* whose lifetime spans *Z-Associations*. However, no metadata is defined to describe this collection. In other words, the collection descriptor for a static persistent unregistered collection is not fully defined.

When a *target* creates a *task package* of type *PersistentResultSet*, a (persistent) *result set* is created, represented by the created *task package*, in the form of a record in the *Extended Services database*. When that *task package* is subsequently retrieved by an *origin*, in either the same or a different *Z-Association*, a

[23] Because of the complexity of creating and maintaining *persistent result sets* for distributed queries, the CIP only supports the use of the *Persistent Result Set ES* for local queries.

copy of that *persistent result set* is made available by the *target* as a Z39.50 *result set* (i.e. as a *transient result set*; a *result set name*, for use during *the Z-Association*, is included within the *task package*).

Table 3-32: Persistent Result Set Extended Service

ASN.1 Definition	Meaning

`PersistentResultSet ::= CHOICE{`
`  esRequest  [1] IMPLICIT SEQUENCE{`
`  toKeep  [1] IMPLICIT NULL,`
`  notToKeep  [2] OriginPartNotToKeep OPTIONAL},`
`  taskPackage  [2] IMPLICIT SEQUENCE{`
`  originPart  [1] IMPLICIT NULL,`
`  targetPart  [2] TargetPart OPTIONAL}}`
	See Section 3.5.7.1.6.
	Note that no parameter supplied by the origin is to be kept by the task package.

`OriginPartNotToKeep ::= SEQUENCE{`
`  originSuppliedResultSet  [1] IMPLICIT InternationalString OPTIONAL,`
`  -- name of transient result set, supplied on`
`  -- request, mandatory unless function is`
`  -- 'delete'`
`  replaceOrAppend  [2] IMPLICIT INTEGER{`
`  -- only if function is "modify"`
`  replace  (1),`
`  append  (2)} OPTIONAL}`
	The **OriginPartNotToKeep** contains the following:
	• **originSuppliedResultSet**, which is the name of a result set created during the current Z-Association and supplied by the origin. Whether this result set is mandatory and the usage of the result set depends on the selected **function**.
	• **replaceOrAppend**, which indicates, when the selected **function** for the ES is modify, whether the result set provided by the origin **originSuppliedResultSet** will replace the existing persistent result set or will be appended to it.

`TargetPart ::= SEQUENCE{`
`  targetSuppliedResultSet  [1] IMPLICIT InternationalString OPTIONAL,`
`  -- Name of transient result set, supplied by`
`  -- target, representing the persistent`
`  -- result set to which package pertains.`
`  -- Meaningful only when package is`
`  -- presented. (i.e. not on ES response)`
`  numberOfRecords  [2] IMPLICIT INTEGER OPTIONAL}`
	The **TargetPart** contains the following:
	• **targetSuppliedResultSet**, which is the name of the copy of the persistent result set represented by the task package when it is returned by the Present Service (but not by an ES response).
	• **numberOfRecords**, which is the number of records that the result set contains.

3.5.7.3 Persistent Query Extended Service

The *Persistent Query Extended Service* allows an *origin* to request that the *target* create a *persistent query*. A *persistent query* is stored persistently (and therefore its lifetime may span *Z-Associations*). Its validity is not restricted to a single *search request*. Further, a *persistent query* may be used over again by any *search request*.

The *Persistent Query ES*, presented in Table 3-33, may be used by the CIP for the promotion of a *result set* to a dynamic persistent unregistered collection (i.e. a collection which content may evolve dynamically over time). In this case, instead of persistently storing the a *result set* for further use, the query (i.e. the set of search criteria) which generated the *result set* may be stored and may be re-invoked at a later time (i.e. during another *Z-Association)*. In this case, of course, the content of the targeted collection may have changed and therefore the *result set* obtained may differ from another generated previously). Note that such a collection is merely a *result set* whose lifetime spans *Z-Associations*. However, no metadata is defined to describe this collection. In other words, the collection descriptor for a dynamic persistent unregistered collection is not fully defined.

When a *target* creates a *task package* of type *PersistentQuery*, a (persistent) Z30.50 *query* is created, represented by the created *task package*, in the form of a record in the *Extended Services database*. When that *task package* is subsequently retrieved by an *origin*, in either the same or a different *Z-Association*, a copy of that *persistent query* is made available by the *target*. However, the *query* is not performed by the *target* until a *Search request* using the *query* is submitted by the *origin*.

Table 3-33: Persistent Query Extended Service

ASN.1 Definition	Meaning

`PersistentQuery ::= CHOICE{`
`  esRequest  [1] IMPLICIT SEQUENCE{`
`  toKeep  [1] OriginPartToKeep OPTIONAL,`
`  notToKeep  [2] OriginPartNotToKeep},`
`  taskPackage  [2] IMPLICIT SEQUENCE{`
`  originPart  [1] OriginPartToKeep OPTIONAL,`
`  targetPart  [2] TargetPart}}`
	See Section 3.5.7.1.6.

`OriginPartToKeep ::= SEQUENCE{`
`  dbNames  [2] IMPLICIT SEQUENCE OF InternationalString ~~OPTIONAL~~,`
`  additionalSearchInfo  [3] OtherInformation OPTIONAL}`
	The **OriginPartToKeep** contains the following:
	• **dbNames**, which contains the name of the collection (i.e. database) at which the persistent query must be targeted.
	For the CIP, the following additional considerations apply:
	  • Only one database name may be provided (i.e. a persistent query, as any CIP query, can be targeted at only one collection).
	  • If a database name is provided, the persistent query is considered specific to a particular collection, and must not be targeted at another collection when used. Otherwise, the persistent query is generic and can be targeted at any collection in the collection hierarchy.
	• **additionalSearchInfo**, see Table 3-6.

`OriginPartNotToKeep ::= CHOICE{`
`  package  [1] IMPLICIT InternationalString,`
`  query  [2] Query}`
	The **OriginPartNotToKeep** contains either:
	• a **package**, which is the name of a task package which contains the definition of an existing persistent query.
	or
	• a **query**, which is the definition of the query that has to be made persistent.

`TargetPart ::= Query`	The **TargetPart** contains the persistent **query**.

3.5.7.4 Periodic Query Schedule Extended Service

The *Periodic Query Schedule Extended Service* allows an *origin* to request that the *target* runs a *persistent query* periodically. Both the *persistent query* and the schedule for its periodic execution are stored persistently (and therefore their lifetime may span *Z-Associations*).

The *Periodic Query Schedule ES*, presented in Table 3-34, may be used by the CIP, for example, for the periodic query of the status information related to an order.

It is important to note that only the originating Retrieval Manager (i.e. the Retrieval Manager which directly receives the *Periodic Query Schedule ES* request) must manage and store the *periodic query schedule*. If a *periodic query* has to be performed across Retrieval Managers, it is the responsibility of the originating Retrieval Manager to manage the sub-queries and to request the appropriate *persistent queries* from other Retrieval Managers).

Table 3-34: Periodic Query Schedule Extended Service

ASN.1 Definition	Meaning

`PeriodicQuerySchedule ::= CHOICE{`
`  esRequest  [1] IMPLICIT SEQUENCE{`
`  toKeep  [1] OriginPartToKeep,`
`  notToKeep  [2] OriginPartNotToKeep},`
`  taskPackage  [2] IMPLICIT SEQUENCE{`
`  originPart  [1] OriginPartToKeep,`
`  targetPart  [2] TargetPart}}`
	See Section 3.5.7.1.6.

`OriginPartToKeep ::= SEQUENCE{`
`  activeFlag  [1] IMPLICIT BOOLEAN,`
`  databaseNames  [2] IMPLICIT SEQUENCE OF InternationalString`
`  OPTIONAL,`
`  resultSetDisposition  [3] IMPLICIT INTEGER{`
`  replace  (1),`
`  append  (2),`
`  createNew  (3)`
`  -- Only if origin and target have agreement`
`  -- about naming convention for the resulting`
`  -- package, and only if no resut set is`
`  -- specified.`
`  } OPTIONAL,`
`  -- Mandatory on 'create' if result set is`
`  -- specified, in which case it must be`
`  -- 'replace' or 'append.`
`  alertDestination  [4] Destination OPTIONAL,`
`  exportParameters  [5] CHOICE{`
`  packageName  [1] IMPLICIT`
`  InternationalString,`
`  exportPackage  [2] ExportSpecification}`
`  OPTIONAL}`
	The **OriginPartToKeep** contains the following:
	• **activeFlag**, which indicates whether the periodic query schedule is activated or not.
	• **databaseNames**, which is the name of the collection that is targeted by the periodic query. Note that for CIP only one database name may be provided. A database name must be specified a periodic query and must therefore be provided if the origin supplies a query rather than a query package name. In the latter case, however, the targeted collection may already be defined in the package, in which case **databaseNames** must be omitted.
	• **resultSetDisposition**, which indicates what the target must do with the result set created by the execution of the periodic query. The following alternatives are possible:
	  • **replace**, which is used to replace the content of the existing result set.
	  • **append**, which is used to append new results at the end of the existing result set.
	  • **createNew**, which is used to create a new result set every time the periodic query is executed.
	• **alertDestination**, which is the destination to which alerts triggered by the receipt of new periodic query results must be sent.
	• **exportParameters**, which is the name, or actual content, of an Export Parameter Package.

ASN.1 Definition	Meaning

`Destination ::= CHOICE{`
`  phoneNumber [1] IMPLICIT InternationalString,`
`  faxNumber [2] IMPLICIT InternationalString,`
`  x400address [3] IMPLICIT InternationalString,`
`  emailAddress [4] IMPLICIT InternationalString,`
`  pagerNumber [5] IMPLICIT InternationalString,`
`  ftpAddress [6] IMPLICIT InternationalString,`
`  ftamAddress [7] IMPLICIT InternationalString,`
`  printerAddress [8] IMPLICIT InternationalString,`
`  other [100] IMPLICIT SEQUENCE{`
`  vehicle [1] IMPLICIT InternationalString OPTIONAL,`
`  destination [2] IMPLICIT InternationalString}}`
	The **Destination** indicates the address to which alerts triggered by the receipt of new periodic query results must be sent. Note however that, as any origin supplied parameters, the target may override, or ignore, the **Destination** suggested by the origin.
	For the CIP, only **phoneNumber**, which is the phone number of the owner of the task package, **emailAddress**, which is the e-mail address of the owner of the task package, and **ftpAddress**, which is the FTP address of the owner of the task package, are considered.

`OriginPartNotToKeep ::= SEQUENCE{`
`  querySpec [1] CHOICE{`
`  actualQuery [1] Query,`
`  packageName [2] IMPLICIT`
`  InternationalString} OPTIONAL,`
`  -- mandatory for 'create'`
`  originSuggestedPeriod [2] Period OPTIONAL, -- mandatory for 'create'`
`  expiration [3] IMPLICIT GeneralizedTime OPTIONAL,`
`  resultSetPackage [4] IMPLICIT InternationalString OPTIONAL}`
	The **OriginPartNotToKeep** contains the following:
	• **querySpec**, which is the definition of the query to be executed periodically. The query can be specified using the following two ways:
	  • **actualQuery**, which is the definition of a query.
	  • **packageName**, which is the name of a Persistent Query Package.
	• **originSuggestedPeriod**, which is the time period between invocations of the periodic query suggested by the origin.
	• **expiration**, which is the date suggested by the origin for the target to discontinue the execution of the periodic query. If not provided, the default assumed by the target is "no expiration".
	• **resultSetPackage**, which is the name of an existing Persistent Result Set package.

ASN.1 Definition	Meaning

`TargetPart ::= SEQUENCE{`
`  actualQuery [1] Query,`
`  targetStatedPeriod [2] Period,`
`  -- Target supplies the period, which may be`
`  -- same as origin proposed.`
`  expiration [3] IMPLICIT GeneralizedTime OPTIONAL,`
`  -- Target supplies value for task package. It`
`  -- may be the same as origin proposed or`
`  -- different from (and overrides) origin`
`  -- proposal, but if omitted, there is no`
`  -- expiration.`
`  resultSetPackage [4] IMPLICIT InternationalString OPTIONAL,`
`  -- May be omitted only if exportParameters was`
`  -- supplied. Target supplies same name as`
`  -- origin supplied, if origin did supply a`
`  -- name.`
`  lastQueryTime [5] IMPLICIT GeneralizedTime,`
`  lastResultNumber [6] IMPLICIT INTEGER,`
`  numberSinceModify [7] IMPLICIT INTEGER OPTIONAL}`
	The **TargetPart** contains the following:
	• **actualQuery**, which is the actual query that is performed periodically. Depending on what was provided by the origin (see **querySpec** in **originPartNotToKeep**) it is either the query definition as supplied by the origin, or a copy of the query defined in the task package supplied by the origin.
	• **targetStatePeriod**, which is the actual time period between invocations of the periodic query. Although the target may in principle override the period supplied by the origin, in the CIP the actual period as performed by the target must be the same as the period suggested by the origin.
	• **expiration**, which is the actual date at which the execution of the periodic query is discontinued by the target. Although the target may in principle override the expiration date supplied by the origin, in the CIP the expiration date considered by the target must be the same as the expiration date suggested by the origin.
	• **resultSetPackage**, which is the name of the task package containing the persistent result set provided by the execution of the periodic query. If the origin provides the name of an existing Persistent Result Set package, that package will be used by the target. Otherwise, the Persistent Result Set package is provided by the target.
	• **lastQueryTime**, which is the time at which the periodic query was last executed.
	• **lastResultNumber**, which indicates the number of new records obtained the last time the periodic query was executed.
	• **numberSinceModify**, which indicates the total number of records obtained through the execution of the periodic query since the last time the Periodic Query package was modified.

`Period ::= CHOICE{`
`  unit [1] IMPLICIT IntUnit,`
`  businessDaily [2] IMPLICIT NULL,`
`  continuous [3] IMPLICIT NULL,`
`  other [4] IMPLICIT InternationalString}`
	Period contains the definition of the time between invocations of the periodic query. The **Period** can be defined as a number of time units, a frequency or as continuous in the following manner:
	• **unit**, i.e. a number of time unit (e.g. number of days, weeks, etc.)
	• **businessDaily**, i.e. every business day.
	• **continuous**, i.e. the periodic query is to be run continuously, or at the target's discretion.
	• **other**, any other definition agreed between the origin and the target.

3.5.7.5 Database Update Extended Service

The *Database Update Extended Service* allows an *origin* to request that the *target* updated one of its *database.*

The *Database Update ES*, presented in Table 3-35, may be used by the CIP to register a persistent unregistered collection as a theme collection.

Table 3-35: Database Update Extended Service

ASN.1 Definition	Meaning

`Update ::= CHOICE{`
`  esRequest  [1] IMPLICIT SEQUENCE{`
`  toKeep  [1] OriginPartToKeep,`
`  notToKeep  [2] OriginPartNotToKeep},`
`  taskPackage  [2] IMPLICIT SEQUENCE{`
`  originPart  [1] OriginPartToKeep,`
`  targetPart  [2] TargetPart}}`
	See Section 3.5.7.1.6.

`OriginPartToKeep ::= SEQUENCE{`
`  action  [1] IMPLICIT INTEGER{`
`  recordInsert  (1),`
`  recordReplace  (2),`
`  recordDelete  (3),`
`  elementUpdate  (4)},`
`  databaseName  [2] IMPLICIT InternationalString,`
`  schema  [3] IMPLICIT OBJECT IDENTIFIER ~~OPTIONAL~~,`
`  elementSetName  [4] IMPLICIT InternationalString OPTIONAL}`
	The **OriginPartToKeep** contains the following:
	• **action**, which indicates the type of operation that is requested to be performed in the database. The supported operations are the following:
	  • **recordInsert**, which is used to insert a new database record.
	  • **recordReplace**, which is used to replace an existing database record.
	  • **recordDelete**, which is used to delete a database record.
	  • **elementUpdate**, which is used to update schema elements within an existing database record.
	• **databaseName**, the name of the database to be updated. As only the CIP collections database may be updated using the Database Update ES, only 'IR-Collection-1' may be used.
	• **schema**, which specifies the applicable database schema.
	• **elementSetName**, which specifies, via an element set name, which schema elements of the updated records are to be kept in the task package. If no element set name is provided, the updated record is not to be included in the task package.

`OriginPartNotToKeep ::= SuppliedRecords`	The **OriginPartNotToKeep** contains the **SuppliedRecords**, i.e. the records that is supplied by the origin to update the database.

ASN.1 Definition	Meaning

`TargetPart ::= SEQUENCE{`
`  updateStatus  [1] IMPLICIT INTEGER{`
`  success (1),`
`  partial (2),`
`  failure (3)},`
`  globalDiagnostics  [2] IMPLICIT SEQUENCE OF DiagRec OPTIONAL,`
`  -- These are non-surrogate diagnostics relating`
`  -- to the task, not to individual records.`
`  taskPackageRecords  [3] IMPLICIT SEQUENCE OF TaskPackageRecordStructure`
`  -- There should be a TaskPackageRecordStructure`
`  -- for every record supplied. The target should`
`  -- create such a structure for every record`
`  -- immediately upon creating the task package`
`  -- to include correlation information and`
`  -- status. The record itself would not be`
`  -- included until processing for that record is`
`  -- complete.`
`  }`
	The **TargetPart** contains the following:
	• **updateStatus**, which indicates the status of the database update operation. The status is one of the following:
	  • **success** indicates that the update has been performed successfully.
	  • **partial** indicates that the update failed for one or more of the records to be updated.
	  • **failure** indicates that the target rejected the execution of the task.
	• **globalDiagnostics**, which contains one or more diagnostics regarding the **failure** of the task.
	• **taskPackageRecords**, which contains the record related data to be returned. Its content depends on both the status of the task and the status of the record.

`SuppliedRecords ::= SEQUENCE OF SEQUENCE{`
`  recordId  [1] CHOICE{`
`  number  [1] IMPLICIT INTEGER,`
`  string  [2] IMPLICIT InternationalString,`
`  opaque  [3] IMPLICIT OCTET STRING} OPTIONAL,`
`  supplementalId  [2] CHOICE{`
`  timeStamp  [1] IMPLICIT GeneralizedTime,`
`  versionNumber  [2] IMPLICIT`
`  InternationalString,`
`  previousVersion  [3] IMPLICIT EXTERNAL`
`  } OPTIONAL,`
`  correlationInfo  [3] IMPLICIT CorrelationInfo OPTIONAL,`
`  record  [4] IMPLICIT EXTERNAL}`
	SuppliedRecords contains the records that are supplied by the origin to update the database. For each record, the following information is supplied:
	• **recordId**, which is the identifier of the database record. The record identifier can be either a **number**, a **string**, or an **opaque** identifier.
	• **supplementalId**, which contains supplemental identification information to allow the target to correctly identify the database record, or the appropriate version of the database record. In the CIP, the supplemental information can be provided in the form of a **timeStamp** or the **versionNumber** of the record.
	• **correlationInfo**, which may be used to identify the record within the context of the update task.
	• **record,** which contains the actual record to be updated.

ASN.1 Definition	Meaning

`TaskPackageRecordStructure ::= SEQUENCE{`
`recordOrSurDiag [1] CHOICE {`
`record [1] IMPLICIT EXTERNAL,`
`-- Choose 'record' if`
`-- recordStatus is 'success',`
`-- and elementSetName was`
`-- supplied.`
`diagnostic [2] DiagRec`
`-- Choose 'diagnostic', if`
`-- RecordStatus is failure.`
`} OPTIONAL,`
`-- The parameter recordOrSurDiag will thus be`
`-- omitted only if 'elementSetName' was omitted`
`-- and recordStatus is 'success'; or if record`
`-- status is 'queued' or in 'process'.`
`correlationInfo [2] IMPLICIT CorrelationInfo OPTIONAL,`
`-- This should be included if it was supplied`
`-- by the origin.`
`recordStatus [3] IMPLICIT INTEGER{`
`success (1),`
`queued (2),`
`inProcess (3),`
`failure (4)}}
	TaskPackageRecordStructure contains the record related data to be returned by the target. The following information is provided:
	• **recordOrSurDiag**, which is provided if the task is complete and contains the actual **record** the **recordStatus** is **success** or a diagnostic in case of **failure**.
	• **correlationInfo**, which may be used to identify the record within the context of the update task.
	• **recordStatus**, which indicates the status of the record. The status may be one of the following:
	  • **success** indicates that the record has been updated successfully.
	  • **queued** indicates that the update of the record is being queued.
	  • **inProcess** indicates that the update of the record is in progress.
	  • **failure** indicates that the update of the record has failed.

`CorrelationInfo ::= SEQUENCE{`
`-- origin may supply one or both for any record:`
`note [1] IMPLICIT InternationalString OPTIONAL,`
`id [2] IMPLICIT INTEGER OPTIONAL}`
	CorrelationInfo may be used in lieu of a record identifier in case a record does not have an unambiguous **recordId**.

3.5.8 Ordering Facility

This section present how Z39.50 Facilities shall be used to perform the ordering of EO data via the CIP. In particular, it explains how the ordering process as described in the Order Technical Note[OTN] can be achieved using the CIP. This section is structured as follows:

- Section 3.5.8.1 describes how the order options available for a particular user shall be retrieved.
- Section 3.5.8.2 describes how an order specification shall be validated and an estimate of the order request shall be returned to the *origin*.
- Section 3.5.8.3 describes how an order specification shall be submitted.
- Section 3.5.8.4 describes how order requests may be monitored by an *origin*.
- Section 3.5.8.5 describes how order requests may be cancelled by an *origin*.
- Section 3.5.8.6 describes how orders may be deleted by an *origin*.
- Section 3.5.8.7 describes how remote orders may be performed by an *origin*.

A detailed description of the dynamic aspects of the use of the order operations supported by the CIP is provided in Appendix G.

3.5.8.1 Order Options Retrieval

Order option definitions are attributes of item descriptors and shall therefore be retrieved in the same manner as other item descriptor attributes, i.e. by the use of the *Present facility*.

Order options (as any kind of options in the CIP) are group based (see also the CIP schemas in Appendix C). This means that a set of order options may be defined for each specific user group, and that therefore different group of users may be assigned different options. This allows, for instance, to define that a group of privileged users user may be delivered products via ftp, whilst all other users are delivered products via mail.

The retrieval of the order options of an item descriptor by a user is performed via a *Present request* and using the 'Opt.' *element set*. The *Present response* will then contain all the orders options of the item descriptor which are defined for the group (or groups) in which the user belongs.[24]

3.5.8.2 Order Validation and Estimation

After an order request has been specified by a user at the *origin*, it can be validated. If the validation process is successful, an approximate and non-binding estimate for the order request shall be returned.

Order validation and estimation is achieved in the CIP via the use of the CIP Order ES, by the definition of an *ES request*. This request either creates a new *task package* (if the order specification is new) or modifies an existing *task package* (if the order specification is a refinement or a correction of a previously sent order specification). In the *task specific parameters*, the request specifies that the action to be performed is validation and estimation, specifies the order request identifier if appropriate, and, most importantly, provides the order specification and the user's personal data as input. Note that, whilst the definition provided as input by the *origin* uses an identical format as the one provided as output by the *target*, the content of the input is essentially unstructured and that all

[24] This means that, in addition to selecting only the *schema elements* pertaining to the 'Opt.' *element set*, the Retrieval Manager will also filter the *retrieval record* so that not all the order options, defined for all the existing groups, are returned to the user, but only the order options for which the user is granted access via group membership.

the *origin* needs to provide is the list of the item descriptors together with preferred order options. The *origin* may also specify how it wants to be informed about the status changes of the request.

The *ES response* will contain either *diagnostics* explaining why the order validation and estimation cannot be performed, or the result of the validation and estimation of the order request, contained in the *task package*. The *target* will provide status information for the order, indicating whether the validation and estimation has been successfully performed or not and, when appropriate, additional information regarding the validation and estimation process. When the validation and estimation is (at least partially) performed, the *target* will also, based on the unstructured order specification provided as input by the *origin*, provide a structured order specification, grouping item descriptors into packages, defining delivery units, and will provide all the necessary order options. The order options set by the *target* will, so far as possible, take into account the preferred options specified by the *origin*, however, the *target* may override any order option suggested by the *origin*.

When a new order specification is provided, the *target* also provides an order identifier that may be used later on to monitor, or cancel, the order request. Note that, as for all operations performed using the Order ES, the *task package* returned by the *target* contains both the order parameters provided as input by the *origin* and the order parameters as returned by the validation and estimation process (which is performed externally). This allows the *origin* to compare the order parameters it supplied (e.g. the order specification to validate) with the order parameters returned by the *target* in order to verify that the changes which might have been performed by the *target* are acceptable to the *origin*.

The data flow for order validation and estimation is illustrated below. Figure 3-10 presents the data flow of the general ES parameters.

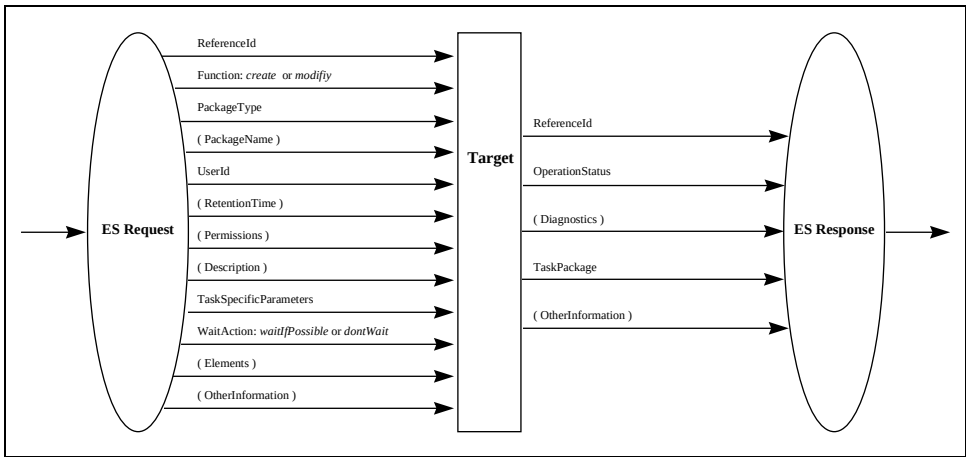

Figure 3-10: Order Validation and Estimation General Parameters Functional Model

Figure 3-11 presents the data flow of the Order ES *task specific parameters*:

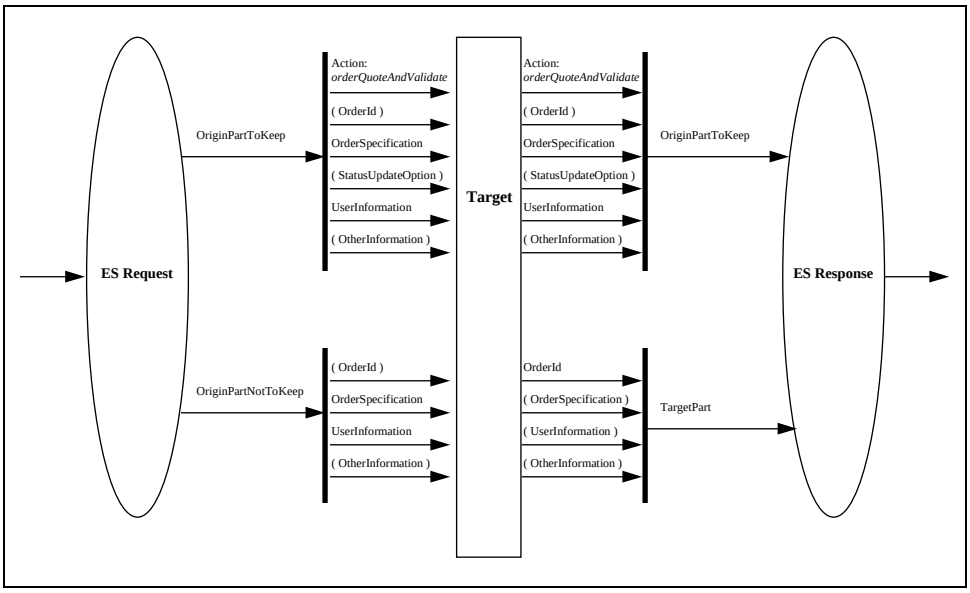

Figure 3-11: Order Validation and Estimation Task Specific Parameters Functional Model

3.5.8.3 Order Submission

After an order request has been specified by a user at the *origin*, and, in principle, after the user has validated the order request and received an estimate (although this is not necessary), the order request can be submitted. If the submission is successful (i.e. the order request is valid), the processing of the order shall be started by the appropriate LOHS.

Order submission is achieved in the CIP via the use of the CIP Order ES, by the definition of an *ES request*. This request may create a new *task package* (if the order specification is new). However, in the vast majority of cases, the request shall modify an existing *task package* containing a previously validated order specification. In the *task specific parameters*, the request specifies that the action to be performed is submission, specifies the order identifier, provides the order specification and all necessary user information as input. The *origin* may also indicates the status update options, i.e. how the user wishes to be informed about the progress of the submission request.

The *ES response* will contain either *diagnostics* explaining why the order submission cannot be performed, or the result of the submission of the order request, contained in the *task package*. The *target* will provide status information for the order, indicating whether the submission has been successfully performed or not and, when appropriate, additional information regarding the submission process. When the submission is successfully performed, the *target* will also include all the precise and binding pricing information in the *task package*.

The data flow for order submission is illustrated below. Figure 3-12 presents the data flow of the general ES parameters.

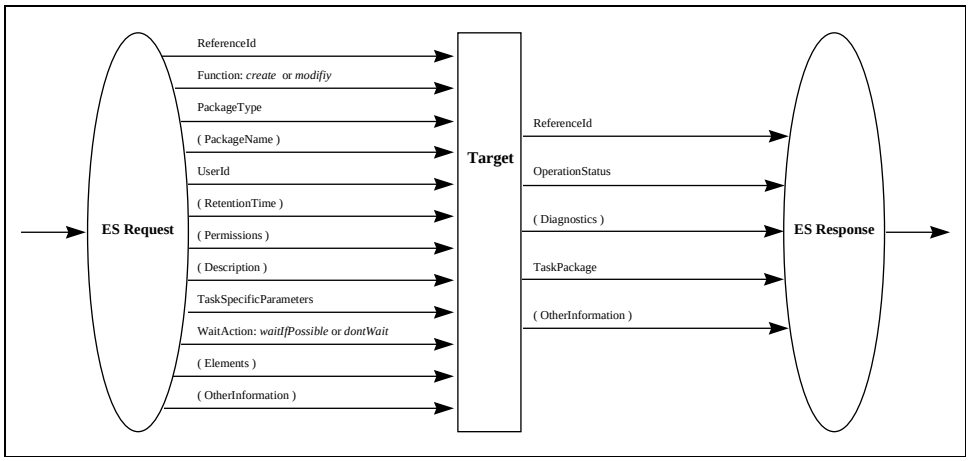

Figure 3-12: Order Submission General Parameters Functional Model

Figure 3-13 presents the data flow of the Order ES *task specific parameters*:

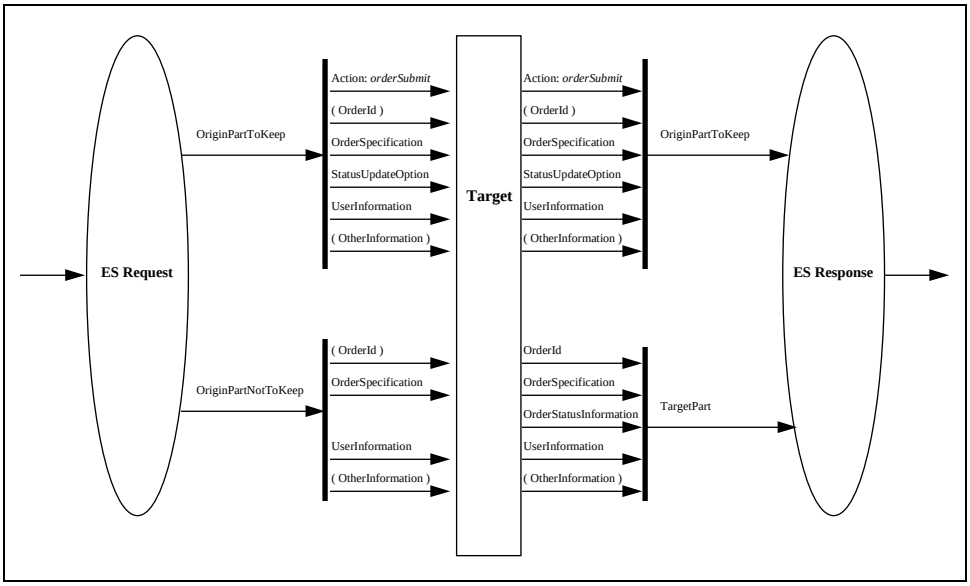

Figure 3-13: Order Submission Task Specific Parameters Functional Model

3.5.8.4 Order Monitoring

After an order request has been submitted, the order request can be monitored by the user. Order monitoring can be performed in following two different methods by the CIP[25]:

- Manual order monitoring, which requires the *origin* to make specific requests to obtain status information. Manual order monitoring via the CIP Order ES is presented in Section 3.5.8.4.1.

[25] The *search* and *retrieval facilities* provided by Z39.50 to query the *ES database*, which contains the *task packages* created by the CIP Order ES, could also be used to monitor order requests. However, the use of this method is discouraged as the information retrieved by this method is not guaranteed to be up to date.
The use of this method by an *origin* can be prevented by the *target* by not granting the *present permission* to the user who owns the *task package* on the *task packages* created by the CIP Order ES. In this way, any attempt to search and retrieve a *task package* created by the CIP Order ES would fail as the user requesting the retrieval would not have the necessary *permissions*.

- Automatic order monitoring, where email is automatically sent as notification of any status change in the processing of the order request by the LOHS. Automatic order monitoring is presented in Section 3.5.8.4.2.

3.5.8.4.1 Manual Order Monitoring

Manual order monitoring is performed via the use of the CIP Order ES with the definition of an ES request and specifying to modify the *task package*. In the *task specific parameters*, the request specifies that the action to be performed is monitoring. Additionally, it may also provide the status update option in order to change the previous setting (i.e. in order to change from manual to automatic order monitoring).

The *ES response* will contain the order identifier and the order status information only.

The data flow for order monitoring is illustrated below. Figure 3-14 presents the data flow of the general *ES parameters*.

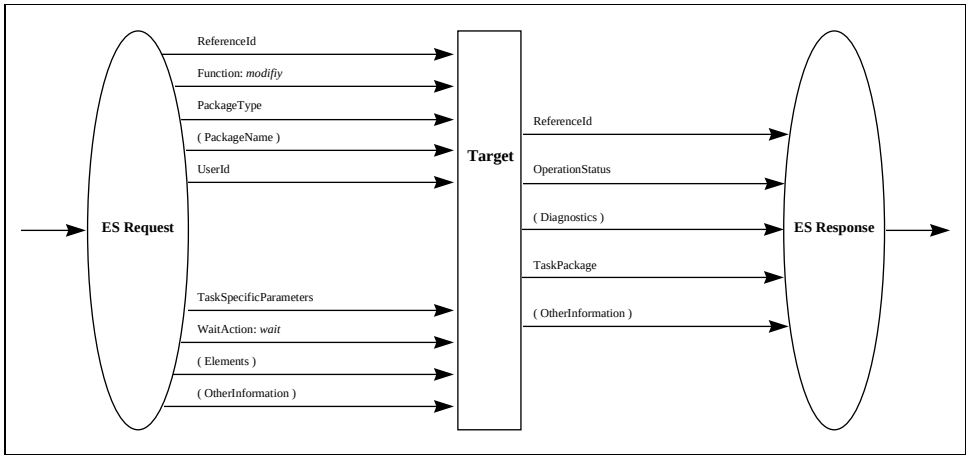

Figure 3-14: Order Monitoring General Parameters Functional Model

Figure 3-15 presents the data flow of the Order ES *task specific parameters*:

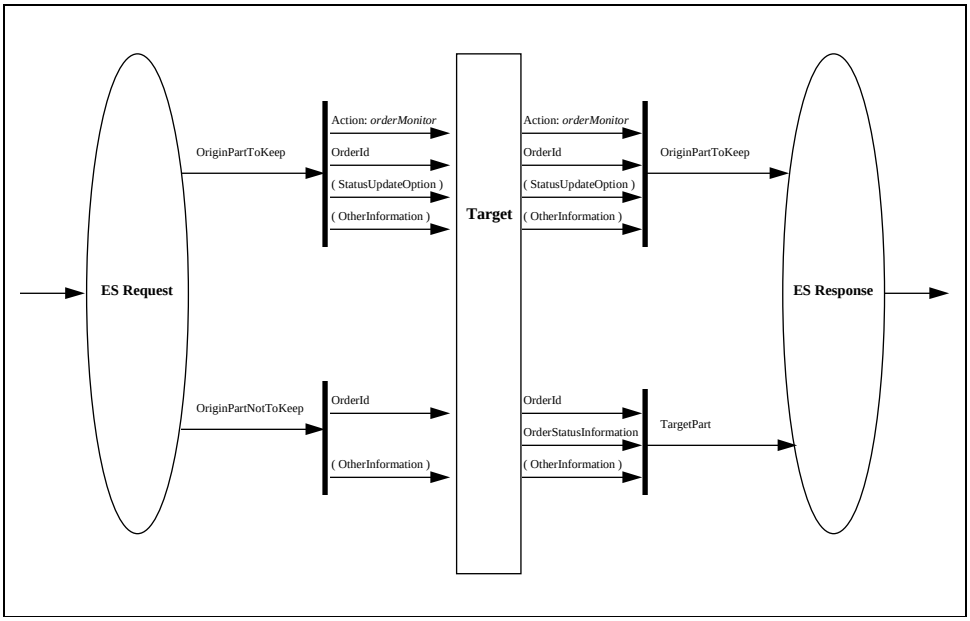

Figure 3-15: Order Monitoring Task Specific Parameters Functional Model

3.5.8.4.2 Automatic Order Monitoring

Automatic order monitoring is supported by the CIP (in that it can be requested by the *target*), but it is performed outside of the CIP. Every time there is a change in the status of the order request an e-mail is sent to the user, warning him of the status change. The user may then use the CIP Order ES to retrieve the order status information included in the *task package*.

3.5.8.5 Order Cancellation

An order request of any type may be cancelled by the *origin*. This does not affect the *task package*. However, it cancels the last order request being performed on the *task package* (i.e.. order estimation, order quotation or order processing). Since the *task package* still exists after the cancellation of an order request, a subsequent order request on the same *task package* may be performed at a later point (e.g. an order may be re-submitted).

Order cancellation is achieved in the CIP via the use of the CIP Order ES, by the definition of an *ES request* and specifying to modify the *task package*. In the *task specific parameters*, the request specifies that the action to be performed is cancellation and specifies the order identifier of the order request to be cancelled.

The *ES response* will contain either a *diagnostic* explaining whether the order cancellation is rejected (e.g. because the cancellation date is already passed) or the result of the cancellation of the order request, contained in the *task package*.

An order can always be cancelled up to the cancellation date (see *orderCancellationDate* in Section 3.5.8.8). But dependent on a specific Retrieval Manager's discretion an order may still be cancelled after the cancellation date.

The data flow for order cancellation is illustrated below. Figure 3-16 presents the data flow of the general ES parameters.

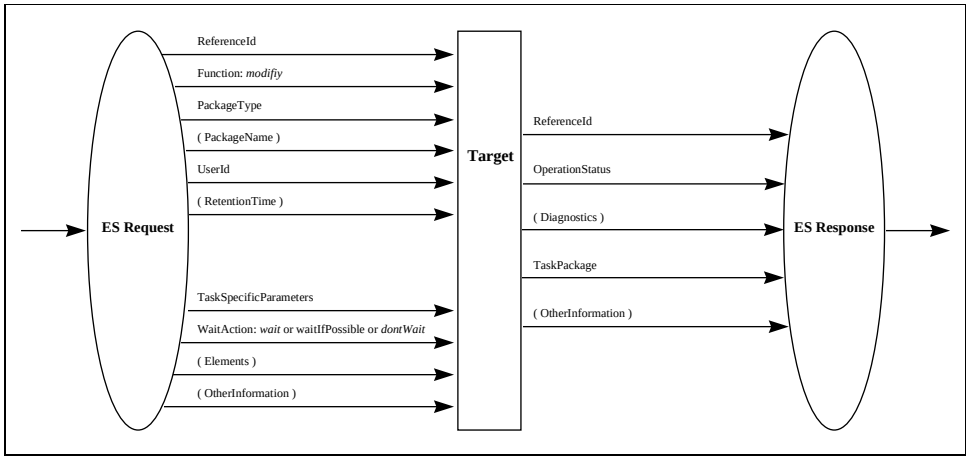

Figure 3-16: Order Cancellation General Parameters Functional Model

Figure 3-17 presents the data flow of the Order ES *task specific parameters*:

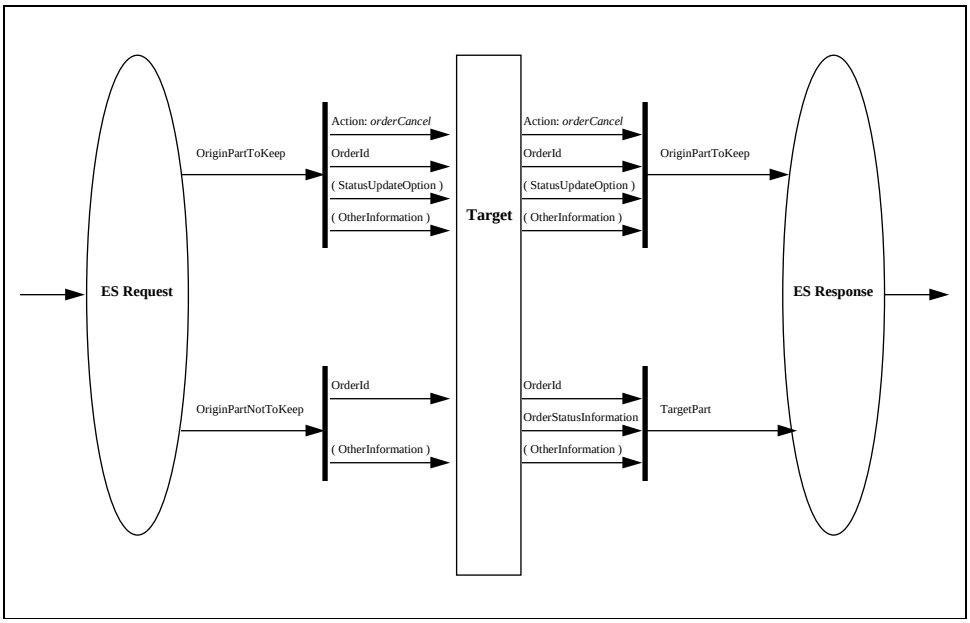

Figure 3-17: Order Cancellation Task Specific Parameters Functional Model

3.5.8.6 Order Deletion

An order request may be deleted by the *origin*, resulting in the deletion of the *task package*.

Order deletion is achieved in the CIP via the use of the CIP Order ES (Section 3.5.8.8), by the definition of an *ES request* and specifying to delete the *task package*. No task specific parameters need to be defined. If the deletion is not successful, the *ES response* will contain a *diagnostic* explaining why the order deletion is rejected (e.g. because the cancellation date of the order is already passed).

An order can be deleted either before the specified cancellation date or after the order has been completed. An order deletion will always automatically cancel an order and remove all order identifiers.

The data flow for order deletion is illustrated below. Figure 3-18 presents the data flow of the general ES parameters (which are the only one required).

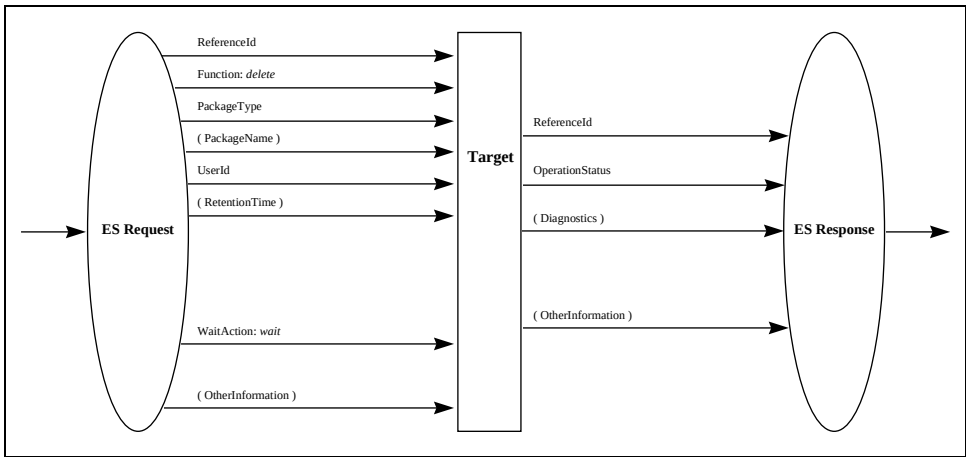

Figure 3-18: Order Deletion Functional Model

3.5.8.7 Remote Ordering

An order request is processed by the Retrieval Manager which directly accesses the local OHS from which the products can be ordered. The local OHS to be accessed for ordering purposes is determined in one of the following two ways:

- The collection path which is returned with the product descriptor in the *Present response* indicates the collection which is the ordering node for the product descriptor (see Section 3.9.4 and Appendix E.7). The ordering node is the first node in the collection path[26] (starting from the collection at which the search was targeted) that contains an OHS identifier.
- The product descriptor contains the identifier OHS(s) from which the product can be ordered.

Two cases may therefore occur when a client performs an order request:

- If the client performs the order request at the Retrieval Manager which directly accesses the local OHS, the RM passes the order request to the local OHS.
- If, on the other hand, the client performs the order request at a Retrieval Manager which does not directly accesses the local OHS, this Retrieval Manager must forward the order request to the remote Retrieval Manager which directly accesses the local OHS (RM B). In this case, RM A acts as an intermediate between the client and RM B. Then RM B can pass the order request to its local OHS as appropriate.

Remote ordering may be performed in the following two ways using the CIP:

- The intermediate Retrieval Manager (RM A) passes the order request to the remote Retrieval Manager (RM B) by proxy for the client. This is presented in Section 3.5.8.7.1.
- The intermediate Retrieval Manager (RM A) is used merely to pass information from the client to the remote Retrieval Manager (RM B). This is presented in Section 3.5.8.7.2.

The choice between proxy remote ordering and pass-through remote ordering is performed by the intermediate Retrieval Manager and is transparent to the client. More details regarding the selection of the type of remote ordering is provided in [SDD].

3.5.8.7.1 Proxy Remote Ordering

In a remote ordering by proxy, the intermediate Retrieval Manager (RM A) performs an order request at a remote Retrieval Manager (RM B) on behalf of a user (via his client) and therefore takes responsibility for the order request[27].

Remote ordering by proxy is achieved in the CIP in the following manner:

- The client performs an Order ES request to RM A. The *ES request* is performed using the user's identifiers, and the *task specific parameters* contain the user's identifier, order specification and personal information.
- Upon reception (and acceptance) of the order request, RM A creates a *task package* for the order request (which is owned by the user). It then performs an Order *ES request* to RM B, forwarding the client's order request to RM B. This *ES request* is performed using RM A's identifier. Moreover, some *task specific parameters* are modified by RM A in order to show that the order request is performed by RM A on behalf of the user rather than directly by the

[26] The collection path may include more than one ordering node.

[27] This implies that the intermediate Retrieval Manager acting as proxy for a user trusts the user (since it takes responsibility for the user) and that a remote Retrieval Manager which accepts an order request by proxy trusts the intermediate Retrieval Manager which submitted the request. In practice, this usually involves that an agreement exists between the different parties.

user. This is achieved by substituting the user's identifier, account and billing information with RM A's identifier, account and billing information. Note however that the rest of the user's information (e.g. delivery address) is not modified since the order is delivered directly to the user.

- Upon reception (and acceptance) of the remote order request, RM B creates a *task package* for the order request (which is owned by RM A).
- When the order request has been performed, RM B sends an Order *ES response* to RM A containing RM A's *task package*.
- Upon reception of this response, RM A updates the user's *task package* and forwards the Order *ES response* back to the client. Note that the user's personal information returned in the *task package* may have been modified by the RM B.

The data flow for remote ordering by proxy is illustrated in Figure 3-19.

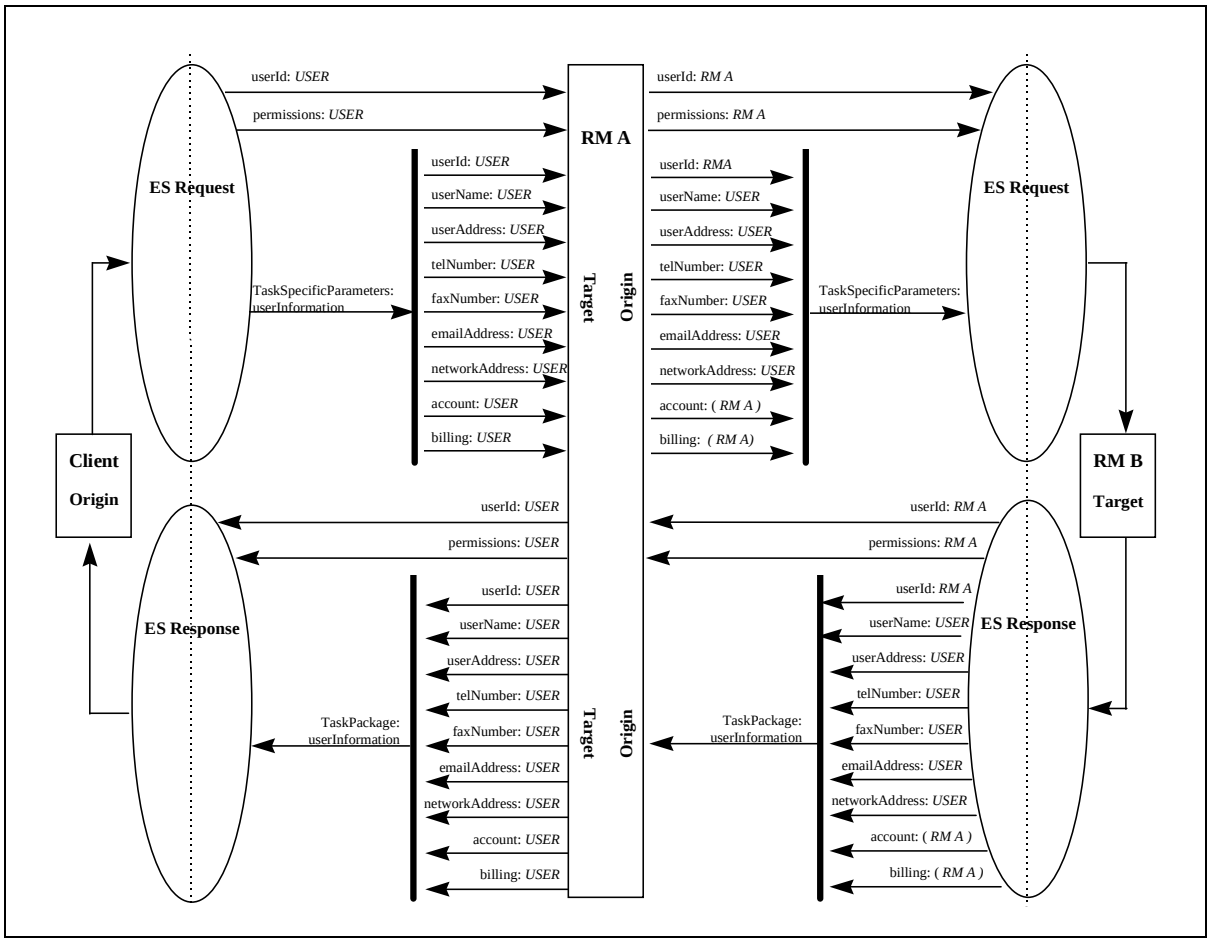

Figure 3-19: Proxy Remote Ordering Functional Model

3.5.8.7.2 Pass-Through Remote Ordering

In a pass-through ordering, the intermediate Retrieval Manager (RM A) simply forwards the order request received from a client to the appropriate remote Retrieval Manager (RM B). RM A does not modify any of the content of the order request and does not take any responsibility for the operation.

Pass-through remote ordering is achieved in the CIP in the following manner:

- The client performs an Order *ES request* to RM A. The *ES request* is performed using the user's identifiers, and the *task specific parameters* contain the user's identifier, order specification and personal information.
- Upon reception (and acceptance) of the order request, RM A creates a *task package* for the order request (which is owned by the user). It then performs an Order *ES request* to RM B, forwarding the client's order request to RM B. This *ES request* is performed using the RM A's identifier. However, RM A does not modify any of the *task specific parameters*.
- Upon reception of the remote order request, RM B checks the identity of the user for whom the remote order request has been forwarded by RM A. Two cases must be distinguished:
  - If RM B has some mutual agreement with RM A regarding the management or recognition of users, RM B is able to recognise the user's identifier provided in the *task specific parameters*. RM B is then able, if necessary[28], to match the user's user identifier at RM A with the user's user identifier that is used in RM B's domain.
    As RM B is confident about the user's identity, RM B is able to verify that the user is allowed to perform ordering with no further checking of the user's identity.
  - Otherwise, RM B cannot be confident about the user's identity[29]. In order to determine the user's identity, RM B sends an *Access Control request* to RM A, requesting the user to provide his user identifier at RM B with the appropriate prompt. This *Access Control request* is then forwarded by RM A to the client so that the user can provide the user identifier by sending back to RM A an *Access Control response* containing the user's user identifier at RM B. RM A then forwards the *Access Control response* back to RM B, which is then able to identify the user and verify that the user is allowed to perform ordering.
- After the remote order request has been accepted, the remote Retrieval Manager creates a *task package* for the order request. This *task package* is owned by RM A. Moreover, some *task specific parameters* are modified by RM B in order to contain the user information recognised by RM B. This is achieved by substituting the user's identifier (which corresponds to the user's identity at RM A) with the user's identifier known at RM B. Additionally, access *permissions* may be granted to the user by RM B in order to allow the user to perform further accesses to the *task package* directly, i.e. without passing through RM A. The rest of the user's information (e.g. delivery address) is not modified.
- When the order request has been performed, the remote Retrieval Manager sends an Order *ES response* to the intermediate Retrieval Manager containing the intermediate Retrieval Manager's *task package*.
- Upon reception of this response, the intermediate Retrieval Manage updates the user's *task package* and forwards the Order *ES response* back to the client. Note that the user's personal information returned in the *task package* may have been modified by the remote Retrieval Manager.

The data flow for pass-through remote ordering is illustrated in Figure 3-20.

[28] This is needed if the two Retrieval Managers, whilst able to recognise the user identifiers managed by the other Retrieval Manager, manage their users independently.

[29] It is possible that the same user identifier is used at two different Retrieval Managers whilst referring to two different persons.

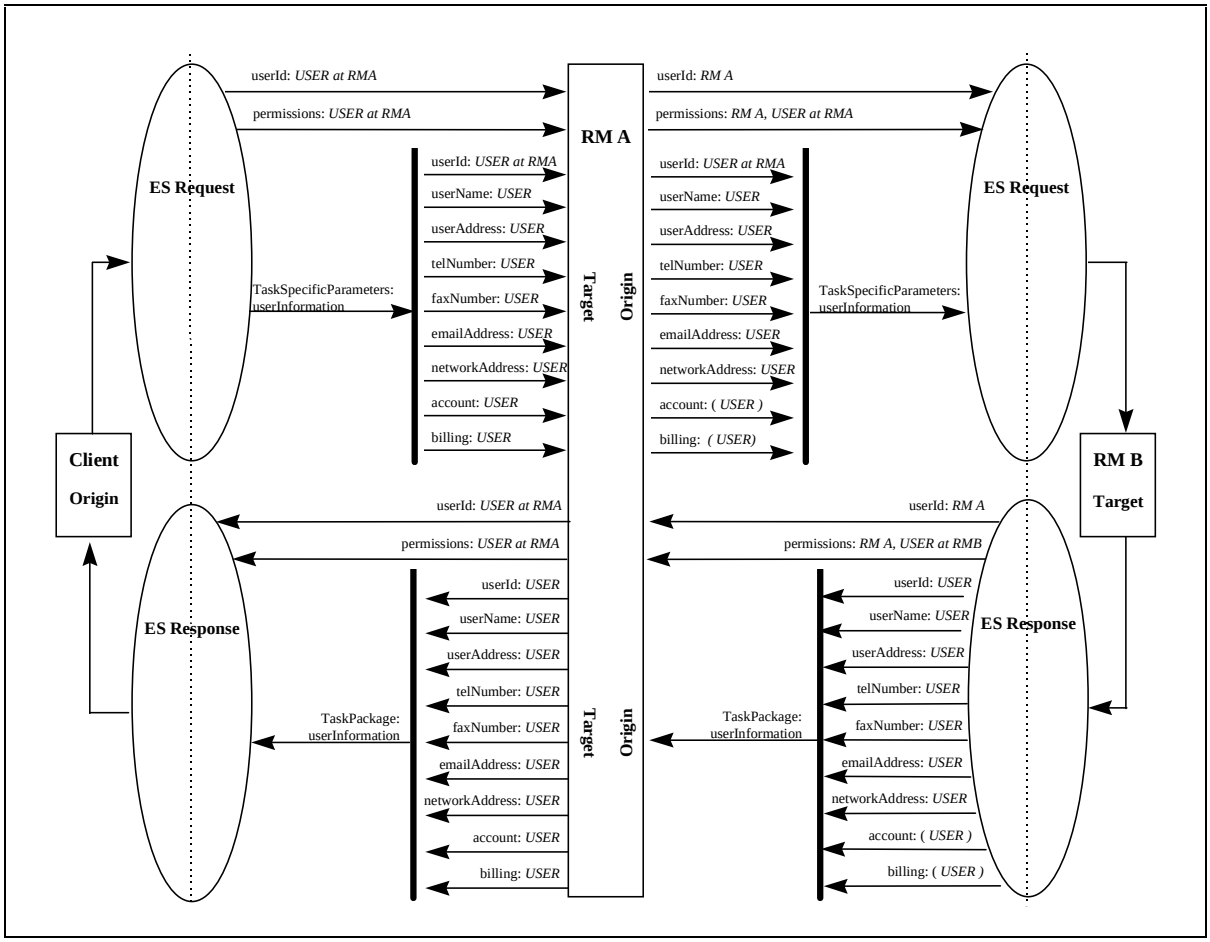

Figure 3-20: Pass-Through Remote Ordering Functional Model

3.5.8.8 CIP Order Extended Service

The *CIP Order Extended Service*, which is a CIP custom ES, allows an *origin* to order products previously queried. The *CIP Order ES* is presented in Table 3-36.

Table 3-36: CIP Order Extended Service

ASN.1 Definition	Meaning

ASN.1 Definition	Meaning

{Z39.50-CIP-Order-ES} DEFINITIONS ::=
BEGIN
IMPORTS OtherInformation, InternationalString, IntUnit
FROM Z39.50-APDU-1995;
 	
CIPOrder ::= CHOICE
{
esRequest [1] IMPLICIT SEQUENCE{
toKeep [1] OriginPartToKeep,
notToKeep [2] OriginPartNotToKeep},
taskPackage [2] IMPLICIT SEQUENCE{
originPart [1] OriginPartToKeep,
targetPart [2] TargetPart}
}
	See Section 3.5.7.1.6.

OriginPartToKeep ::= SEQUENCE
{
action [1] IMPLICIT INTEGER {
orderEstimate (1),
orderQuoteAndSubmit (2),
orderMonitor (3),
orderCancel (4)},
orderId [2] InternationalString OPTIONAL,
orderSpecification [3] OrderSpecification OPTIONAL,
statusUpdateOption [4] StatusUpdateOption OPTIONAL,
userInformation [5] UserInformation OPTIONAL,
otherInfo [6] OtherInformation OPTIONAL
}
	The **OriginPartToKeep** contains the following:
	• **action**, which indicates the type of operation that is requested to be performed for the order request. The supported operations are the following:
	  • **orderEstimate**, which is used to validate and obtain the estimate of an order specification.
	  • **orderQuoteAndSubmit**, which is used to quote[30] and submit an order specification.
	  • **orderMonitor**, which is used to monitor the progress of the processing of an order request.
	  • **orderCancel**, which is used to cancel an order request.
	• **orderId**, which is the identifier of the order request as provided as input by the origin.
	• **orderSpecification**, which is the specification of the order request as provided as input by the origin.
	Note that, in principle, the order request specified by the origin is unstructured, i.e. it contains a list of item descriptor identifiers and the order options related to them, but does not attempt to group them into packages and delivery units.
	• **statusUpdateOption**, which indicates how the origin wishes to be kept up to date as to the status of the order processing.
	• **userInformation**, which contains the personal user information as provided as input by the origin.
	• **otherInformation**, which contains additional information not specified by the CIP.

[30] The estimate for an order is approximate and non-binding, whereas the quote for an order is precise and binding.

ASN.1 Definition	Meaning

OriginPartNotToKeep ::= SEQUENCE
{
orderId [1] InternationalString OPTIONAL,
orderSpecification [2] OrderSpecification OPTIONAL,
userInformation [3] UserInformation OPTIONAL,
otherInfo [4] OtherInformation OPTIONAL
}
	The **OriginPartNotToKeep**[31] contains the following:
	• **orderId**, which is the identifier of the order request.
	• **orderSpecification**, which is the specification of the order request.
	• **userInformation**, which contains the personal user information.
	• **otherInformation**, which contains additional information not specified by the CIP.

TargetPart ::= SEQUENCE
{
orderId [1] InternationalString,
orderSpecification [2] OrderSpecification OPTIONAL,
orderStatusInfo [3] OrderStatusInfo OPTIONAL,
userInformation [4] UserInformation OPTIONAL,
otherInfo [5] OtherInformation OPTIONAL
}
	The **TargetPart** contains the following:
	• **orderId**, which is the identifier of the order request as provided as output by the target.
	• **orderSpecification**, which is the specification of the order request as provided as output by the target. This order specification provided by the target overrides the specification provided as input by the origin in **originPartNotToKeep.** It contains the item descriptors and order options supplied as input, with any necessary modifications or additions, in a structured manner, i.e. the item descriptors are grouped into packages and delivery units.
	• **orderStatusInfo**, which indicates the status of the order request being performed[32].
	• **userInformation**, which contains the personal user information.
	• **otherInfo**, which contains additional information not specified by the CIP

StatusUpdateOption ::= CHOICE
{
manual [1] NULL,
automatic [2] IMPLICIT INTEGER {
eMail (1)}
}
	The **StatusUpdateOption** provides options for how the user will receive updates on the status of an **extended service** request. The parameters are:
	• **manual** the user performs the status request.
	• **automatic** where the OHS filing the order provides status updates for the user via **email**[33].

[31] The definitions used in *OriginPartNotToKeep* are strictly identical to the ones provided in *OriginPartToKeep*. The former is used as input by the *target* (which may overwrite some values as appropriate) for the definition of *TargetPart*, whereas the latter remains unmodified and is stored in the *task package*. This duplication therefore allows the comparison of the order as specified by the *origin* (*OriginPartToKeep*) with the order as returned by the *target* (*TargetPart*).

[32] Note the difference between the *operationStatus*, which is provided in the *ES Response*, and the *orderStatusInfo*, which is included in the *task package*. *operationStatus* (see Section 3.5.7.1.4) provides status information for the *ES operation* as a whole and indicates whether the *ES operation* has been performed successfully or not by the *target*. *orderStatusInfo* provides status information for the order specified in the *task package* and indicates the state of the order or the process being performed for an order at the LOHS.

[33] This could be expanded in the future to include, for example, automatic update via the *origin*.

ASN.1 Definition	Meaning

`UserInformation ::= SEQUENCE`
`{`
`userId [1] InternationalString,`
`userName [2] InternationalString OPTIONAL,`
`userAddress [3] PostalAddress OPTIONAL,`
`telNumber [4] InternationalString OPTIONAL,`
`faxNumber [5] InternationalString OPTIONAL,`
`emailAddress [6] InternationalString OPTIONAL,`
`networkAddress [7] InternationalString OPTIONAL,`
`billing [8] Billing OPTIONAL`
`}`
	The **Userinformation** structure is presented by the origin part of a request to a target. The information provided contains mandatory fields (the user identifier) and optional fields. The target will allow the **Userinformation** structure contents to be used as an input to the delivery specification for elements which can be altered by the user. The target will refer to the local database contents for the user and will use the contents of the database, or the **Userinformation** structure depending on the privilege of the user to offer alternative information. The **UserInformation** structure consists of the following attributes:
	• **userId** the user identifier, the identifier which the user provides as part of an **InitializeRequest.**
	• **userName** the full name of the user.
	• **userAddress** a structure to hold the users address.
	• **telNumber** the users telephone number.
	• **faxNumber** the fax number for the user.
	• **emailAddress** the electronic mail address for the user.
	• **networkAddress** the network address to send files to electronically. For Internet addresses, the address is written in URL format to allow directories as well as domain's to be specified.
	• **billing** the method of payment (and hence of billing) available for the user.

`OrderSpecification ::= SEQUENCE`
`{`
`orderingCentreId [1] InternationalString,`
`orderPrice [2] PriceInfo OPTIONAL,`
`orderDeliveryDate [3] InternationalString OPTIONAL,`
`orderCancellationDate [4] InternationalString OPTIONAL,`
`deliveryUnits [5] SEQUENCE OF DeliveryUnitSpec,`
`otherInfo [6] OtherInformation OPTIONAL`
`}`
	The **OrderSpecification** is the specification of the order request and contains the following:
	• **orderingCentreId**, which identifies the ordering centre at which the order will be performed.
	• **orderPrice**, which is the price for the whole order.
	• **orderDeliveryDate**, which is the latest date at which the order can be expected to be delivered to the user.
	• **orderCancellationDate**, which is the latest date at which the user can cancel the order.
	• **deliveryUnits**, which contains the definition of the delivery units which compose the order.
	• **otherInfo**, which may be used to provide additional information not specified by the CIP.

ASN.1 Definition	Meaning

`DeliveryUnitSpec ::= SEQUENCE`
`{`
`deliveryUnitId [1] InternationalString OPTIONAL,`
`deliveryUnitPrice [2] PriceInfo OPTIONAL,`
`deliveryMethod [3] DeliveryMethod OPTIONAL,`
`billing [4] Billing OPTIONAL,`
`packages [5] SEQUENCE OF PackageSpec,`
`otherInfo [6] OtherInformation OPTIONAL`
`}`
	The **DeliveryUnitSpec** contains the specification of a single delivery unit (i.e. part of an order that is delivered as a unit):
	• **deliveryUnitId**, which is the identifier of the delivery unit.
	• **deliveryUnitPrice**, which is the price of the delivery unit.
	• **deliveryMethod**, which is the method with which the delivery unit is delivered to the user.
	• **billing**, which is the method with which the user is going to be billed.
	• **packages**, which contains the definition of the packages which compose the delivery unit.
	• **otherInfo**, which may be used to provide additional information not specified by the CIP.

`DeliveryMethod ::= CHOICE`
`{`
`eMail [1] InternationalString,`
`ftp [2] FTPDelivery,`
`mail [3] PostalAddress,`
`otherInfo [4] OtherInformation`
`}`
	The **DeliveryMethod** defines the method with which a delivery unit is delivered to the user and is one of the following:
	• **eMail**, which specifies the email address that the order will be delivered to
	• **ftp**, which specifies that the order will be delivered via ftp, the type of transfer and the ftp address
	• **mail**, which specifies that the order will be delivered via mail and provides the postal address
	• **otherInfo**, which may be used to provide additional information (such as an alternative delivery method) not specified by the CIP.

ASN.1 Definition	Meaning

FTPDelivery ::= SEQUENCE
{
transferDirection [1] IMPLICIT INTEGER
{
push (0),
pull (1)
},
ftpAddress [2] InternationalString
}
	The **FTPMethod** defines the method with which a delivery unit is delivered to the user and is one of the following:
	• **transferDirection**, which specifies that the order will be delivered via e-mail.
	• **ftpAddress**, which specifies that the order will be delivered via ftp.

Billing ::= SEQUENCE
{
paymentMethod [1] PaymentMethod,
customerReference [2] IMPLICIT CustomerReference,
customerPONumber [3] IMPLICIT InternationalString OPTIONAL
}
	The **Billing** structure[34] contains attributes which describe the method by which a user will pay for a service, together with supporting information regarding the payment. The attributes are:
	• **paymentMethod** indicates the method of payment used.
	• **customerReference** is the customer provided reference for the order.
	• **customerPONumber** is the purchase order provided by the customer for the order.

PaymentMethod ::= CHOICE
{
billInvoice [0] IMPLICIT NULL,
prepay [1] IMPLICIT NULL,
depositAccount [2] IMPLICIT NULL,
privateKnown [3] IMPLICIT NULL,
privateNotKnown [4] IMPLICIT EXTERNAL},
}
	The **PaymentMethod** structure contains attributes which describe the method by which a user will pay for a service. The attributes are:
	• **billInvoice** indicates that an invoice is to be sent to the user (or payee).
	• **prepay** indicates that payment has already been agreed/performed.
	• **depositAccount** indicates that there is a deposit account for the payment.
	• **privateKnown** indicates that the payment method is private and known.
	• **privateNotKnown** contain private unknown payment method information.

CustomerReference ::= SEQUENCE
{
customerId [1] InternationalString,
accounts [2] SEQUENCE OF InternationalString
}
	The **CustomerReference** structure contains attributes which provide a customer reference for the order. The attributes are:
	• **customerId** indicates the customer identifier at the LOHS.
	• **accounts** is the name of the account(s) available to apply charges to on behalf of the user.

PostalAddress ::= SEQUENCE
{
streetAddress [1] InternationalString,
city [2] InternationalString,
state [3] InternationalString,
postalCode [4] InternationalString,
country [5] InternationalString
}
	PostalAddress contains the postal address for a user and consists of:
	• **streetAddress**, which is the street name and number.
	• **city**, which is the name of the city (or nearest city).
	• **state**, which is the name of the state or county.
	• **postalCode**, which is the country specific postal code.
	• **country**, which is the name of the country.

[34] The Billing structure used by the CIP is derived from the *addlBilling* structure defined in the *Item Order ES*.

ASN.1 Definition	Meaning

PackageSpec ::= SEQUENCE
{
packageId [1] InternationalString OPTIONAL,
packagePrice [2] PriceInfo OPTIONAL,
package [3] CHOICE
{
predefinedPackage [1] PredefinedPackage,
adHocPackage [2] AdHocPackage
},
packageMedium [4] InternationalString,
packageKByteSize [5] INTEGER,
otherInfo [6] OtherInformation OPTIONAL
}
	The **PackageSpec** contains the specification of a single package (i.e. part of an order that is delivered on a single medium):
	• **packageId**, which is the identifier of the package.
	• **packagePrice**, which is the price of the package.
	• **package**, which contains the specification of the package. The package is one of the following:
	  • **predefinedPackage**, which is a package pre-defined by the data provider.
	  • **adHocPackage**, which is a package constructed ad-hoc by the data provider to fulfil the order request.
	• **packageMedium**, which is the medium on which the package will be delivered to the user.
	• **packageKByteSize**, which contains the size of the **package** in kilobytes.
	• **otherInfo**, which may be used to provide additional information not specified by the CIP.

PredefinedPackage ::= SEQUENCE
{
collectionId [1] InternationalString,
orderItems [2] SEQUENCE OF OrderItem,
otherInfo [3] OtherInformation OPTIONAL
}
	A **PredefinedPackage** contains the definition of a package that is pre-defined by the data provider. A PredefinedPackage is a collection that is stored in advance (i.e. not to fulfil a specific order) on a medium and is defined as follows:
	• **collectionId**, which is the identifier of the pre-packaged collection. Must be formatted according to the naming convention for collection identifiers specified in Appendix E.
	• **orderItems**, which contains the list of the order items contained in the package.
	• **otherInfo**, which may be used to provide additional information not specified by the CIP.

ASN.1 Definition	Meaning
`AdHocPackage ::= SEQUENCE OF OrderItem`	An **AdHocPackage** is a package that is defined ad-hoc by a data provider to fulfil a specific order. An **AdHocPackage** contains the list of the order items contained in the package.
`OrderItem ::= SEQUENCE { productId [1] InternationalString, productPrice [2] PriceInfo OPTIONAL, productDeliveryOptions [3] ProductDeliveryOptions OPTIONAL, processingOptions [5] ProcessingOptions OPTIONAL, sceneSelectionOptions [6] SceneSelectionOptions OPTIONAL, orderStatusInfo [7] OrderStatusInfo OPTIONAL, otherInfo [8] OtherInformation OPTIONAL }`	The **OrderItem** contains the specification of a single order item (i.e. the product that is ordered and that is to be delivered): • **productId**, which is the identifier of the ordered product. • **productPrice**, which is the price of the product. • **productDeliveryOptions**, which contains delivery options for the product. • **processingOptions**, which specifies the processing options that are to be applied on the product before delivery. • **sceneSelectionOptions**, which specifies the selection of the scene from the whole product that is to be delivered. • **orderStatusInfo**, which indicates the status of the order item[35]. • **otherInfo**, which may be used to provide additional information not specified by the CIP.
`ProductDeliveryOptions ::= SEQUENCE { productByteSize [1] INTEGER OPTIONAL, productFormat [2] InternationalString OPTIONAL, productCompression [3] InternationalString OPTIONAL, otherInfo [4] OtherInformation OPTIONAL }`	The **ProductDeliveryOptions** contains the specification of the options regarding the delivery of a product: • **productByteSize**, which contains the size of the product in bytes. • **productFormat**, which specifies the format of the product. • **productCompression**, which specifies the compression mechanism applied to the product. • **otherInfo**, which may be used to provide additional information not specified by the CIP.
`ProcessingOptions ::= CHOICE { formattedProcessingOptions [1] EXTERNAL, unformattedProcessingOptions [2] InternationalString }`	The **ProcessingOptions** specifies the processing options that are to be applied on the product before delivery and is one of the following: • **formattedProcessingOptions**, which specifies the processing options according to the format specified in [ORD]. • **unformattedProcessingOptions**, which specifies the processing options in a free-text form.

[35] Note the difference between the *orderStatusInfo* in *TargetPart*, which indicates the state, or the process being performed for, an order as a whole at the LOHS, and the *orderStatusInfo* in *OrderItem*, which indicates the state, or the process being performed for, a specific order item within an order at the LOHS.

ASN.1 Definition	Meaning

SceneSelectionOptions ::= CHOICE
{
formattedSceneSelectionOptions [1] EXTERNAL,
unformattedSceneSelectionOptions [2] InternationalString
}
	The **SceneSelectionOptions** specifies the selection of the scene from the whole product that is to be delivered and is one of the following:
	• **formattedSceneSelectionOptions**, which specifies the scene selection options according to the format specified in [ORD].
	• **unformattedSceneSelectionOptions**, which specifies the scene selection options in a free-text form.

PriceInfo ::= SEQUENCE
{
price [1] IntUnit,
priceExpirationDate [2] InternationalString,
additionalPriceInfo [3] InternationalString OPTIONAL
}
	The **PriceInfo** contains the information related to the price of an item:
	• **price**, which contains the price of the item.
	• **priceExpirationDate**, which specifies the latest date at which the price provided is valid (i.e. until the expiration date the origin is guaranteed that the price will not vary. However, after the expiration date the price may change).
	• **additionalPriceInfo**, which may be used to provide a textual explanation when the price of a item differs from the sum of the elements which compose this item (e.g. it can be used to explain why the price of a delivery unit differs from the sum of the prices of the packages which compose the delivery unit).

OrderStatusInfo ::= SEQUENCE
{
orderState [1] CHOICE
{
staticState [1] StaticState,
dynamicState [2] DynamicState
},
additionalStatusInfo [2] InternationalString OPTIONAL
}
	OrderStatusInfo describes the status of an extended service order request. The different status values are:
	• **orderState** indicates the state of the order request or the processing being performed for the order:
	  • **staticState** indicates the state of the order when no order request is being performed.
	  • **dynamicState** indicates the processing that is currently performed for an order request.
	• **additionalStatusInfo** contains additional status information provided by the LOHS (e.g. to clarify the meaning of the **orderState**).

StaticState ::= [1] IMPLICIT INTEGER
{
orderNotValid (1),
orderEstimated (2),
orderCompleted (3)
}
	StaticState describes the state of an order when no order request is active. The possible states are:
	• **orderNotValid** indicates that the order has not been successfully validated.
	• **orderEstimated** indicates that the order has been successfully validated and that an estimate is provided.
	• **orderCompleted** indicates that the order has been completed.

ASN.1 Definition	Meaning

DynamicState ::= [2] IMPLICIT INTEGER
{
orderBeingEstimated (4),
orderBeingQuoted (5),
orderBeingProcessed (6),
orderBeingCancelled (7),
orderBeingDeleted (8)
}
 	
END
	DynamicState describes the state of an order when an order request is active and thus being process. The possible states are:
	• **orderBeingEstimated** the order is currently being estimated by the target order handling system.
	• **orderBeingQuoted** the order is currently being quoted by the target order handling system.
	• **orderBeingProcessed** the order is currently being processed by the target order handling system.
	• **orderBeingCancelled** the order request which was previously sent to the target is being cancelled.
	• **orderBeingDeleted** the order is being deleted.

3.5.9 Explain Facility

3.5.9.1 Introduction

The *Explain facility* has been developed as part of Z39.50 primarily to enable the capabilities of a *target* server to be ascertained by clients. The Z39.50 *Explain facility* is an important service for the CIP and one which enables many requirements associated with location, retrieval, manipulation and negotiation of CIP data types to be supported.

The *Explain facility* is also a key tool in the support of interoperability between different clients and servers in the CIP domain.

Within the CIP domain, the Retrieval Manager is an entity that has three primary components; the *Explain database*, a Z39.50 server and the core software components that implement the Retrieval Manager application software. For simplicity, references to the Retrieval Manager are assumed to incorporate these components.

Note that logically the collection database is also part of the Retrieval Manager, i.e. the collection nodes, collection hierarchy etc., which the Retrieval Manager uses to determine collection positions and to provide hierarchy information for hierarchical searches. The collection database is not however shown as a separate entity on the diagram as its content is dependent on the local site.

Z39.50 terminology refers to *origins* and *targets*. The CIP analogies for *origins* and *targets* are clients and Retrieval Managers. In this way, *Explain* queries are targeted at Retrieval Managers and thus at *Explain databases*.

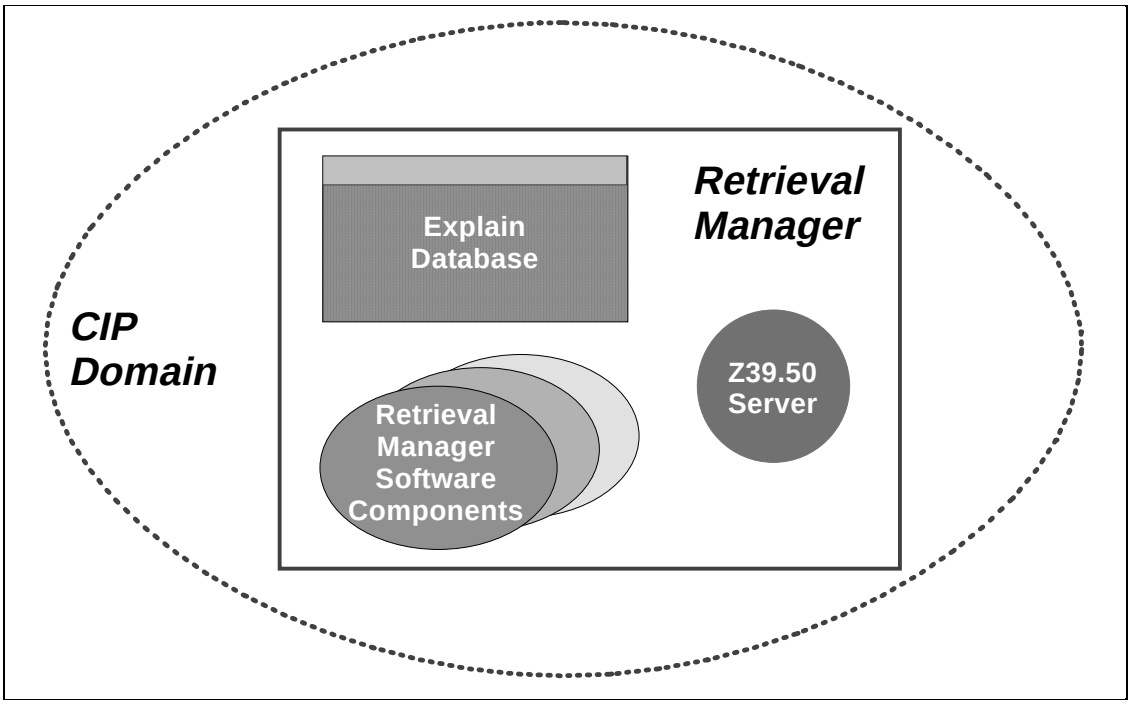

Figure 3-21: Explain Database and Retrieval Manager

In the CIP domain *Explain databases* will therefore be accessed by Retrieval Managers to support various mechanisms including:

- dynamic configuration of clients to match Retrieval Manager controlled data;
- provision of *attribute* and *element* information for use in the CIP;

- provision of *attribute* and *element* information for visibility to the end user;
- provision of general Retrieval Manager information.

This *Explain* section of the CIP specification contains the following:

- structures and implementation of CIP *Explain databases* according to *Explain information categories*;
- the mechanisms by which the *Explain facility* services support the CIP;
- summaries of the different *Explain* search methods.

There are seventeen categories of *Explain information*, each of which provides different information about the catalogue owned by the Retrieval Manager. Each category can be implemented separately and unsupported categories can be added later as needed.

It is important to note that for CIP Release B not all the *Explain categories* are required, but others may be added as required e.g. to support Release C functionality.

Although the *Explain database* is effectively part of the Retrieval Manager, it is possible that a site could maintain additional categories and information which can be retrieved directly by the client without the use of the CIP and the Retrieval Manager, but this is outside the scope of this specification.

The categories supported at any particular *target* can be found out by accessing the ***CategoryList*** *Explain record*. As the *target* capabilities are negotiated during the ***Init*** service, even if a client or server does not support a particular capability, interoperability is still guaranteed.

The following subsections describe the Z39.50 *Explain categories*, and elements within those categories that are required for CIP Release B. The objective is to ensure that any optional elements in Z39.50 that are mandatory for the CIP are highlighted for inclusion and also to indicate any Z39.50 elements that do not require inclusion for the CIP. (Note that a CIP site is free to include any CIP optional elements, e.g. for the provision of extra information to the user).

3.5.9.2 Explain Database Information Categories

The *Explain information* is held in an *Explain database*, which appears to *target* (Retrieval Manager) as another *target database* available for searching and retrieval. The *Explain facility* enables clients to obtain information controlled by the *target* such as available *databases*, *attribute sets*, *diagnostic sets*, *schema, record syntax* and *element specifications*.

A Retrieval Manager controlling an *Explain database* provides the following:

- access to the *Explain database - IR-Explain-1*;
- support to the *Explain attribute set - exp-1*, which defines a set of *Use attributes* to search the *Explain database* and imports *bib-1 non-Use attributes*;
- support of the *Explain syntax*, defined and registered in [Z3950], Appendix REC.1.

The *Explain database* manages three kinds of information:

- General information about a *target* - the form of which will be heterogeneous across Retrieval Managers (although not the details);
- CIP data information - used to aid the client/server negotiation;
- Target database detailed information that is likely to vary between Retrieval Managers, e.g. CIP product descriptor attributes.

This subsection summarises the different types of *Explain information categories*, as detailed in Section 3.2.10.3 of the Z39.50 specification[Z3950] and highlights the subset of the categories that is supported in CIP Release B:

- *Target* server general information (detailed information is provided in Section 3.5.9.2.1)
  - *TargetInfo* - Information about the *target* and its services and constraints, e.g. hours of operation of the *target*, whether the *target* supports named *result sets* and what the maximum size of a *result set* is.
  - *ExtendedServicesInfo* - Description of each *extended service* supported by the *target*, e.g. what restrictions or fees apply for the *extended service* and a human readable description of the *extended service.*
  - *CategoryList* - lists the *explain categories* that are supported by the *target*. In the case of the CIP, the *CategoryList* will only contain the *explain categories* that are supported.
- CIP data information (a detailed description is provided in Section 3.5.9.2.2).
  - *RecordSyntaxInfo* - Information about each *record syntax* supported by the *target* (e.g. a human readable description of an abstract *record syntax)*.
  - *AttributeSetInfo* - Description of each *attribute set* supported by the *target*.
  - *UnitInfo* - Descriptive information about the unit systems supported by the *target*, e.g. the name of each unit system and a list of their unit types.
  - *TagSetInfo* - Information about a given *tag set* (e.g. the name and object identifier of the *tag set*, a detailed description of the elements the *tag set* consists of).
  - *SchemaInfo* - Description of each *database schema*, e.g. the object identifier and the *tag sets* used by the *schema.*
- Target Database Information (detailed information is provided in Section 3.5.9.2.3).
  - *DatabaseInfo* - Detailed description of a *database* and its restrictions and parameters, e.g. the number of records available in a *database*. In the CIP collections are treated as *databases*. For each collection a *DatabaseInfo Explain record* is defined providing information about the *schemas*, *record syntaxes*, *attribute sets*, etc. available for this particular collection.
  - *AttributeDetails* - Information about each *attribute*, e.g. name of the *database* to which this *attribute* information applies and an identifier of the *attribute set* to which the *attribute* belongs to. For each *attribute type* (e.g. *Use attribute*, *Relation attribute*) a list of its possible values are given.
  - *TermListInfo* - Descriptive information about *term-lists*. In the CIP, this *Explain record* is used to list for a *database* (collection) all the lists of valids that are available for this specific *database*.
  - *TermListDetails* - This provides descriptive information for a *term-list*. In the CIP, *TermListDetails* is used to provide information about a list of valids (a *term list*).The information provided includes the *term list*'s name, the link of the *term list* to its relevant *use attribute* (via *attribute combinations*) and the valids that are part of this *term list* (*sampleTerms*).
  - *ElementSetDetails* - Descriptive information about an *element set*, e.g. the *database* to which the *record* pertains.
  - *RetrievalRecordDetails* - Information about the elements of a *retrieval record*, e.g. the *database*, *schema*, and *record syntax* to which the *record* pertains.

A number of the *explain information categories* are not required for this release of the CIP. These *categories* are only briefly listed below for completeness:

- CIP Data Information
  - *VariantSetInfo* - Descriptive information about a *variant set* definition supported by the *target*; classes, types, and values supported for a particular *variant set*. A *variant* defines one of several forms in which an *element* is available for retrieval. The *origin* may *request*, or the *target present*, an *element* according to a specific *variant*. A *variant set* is a definition of a set of classes, for each class, a set of types and for each type a set of values.
- Target Database Information
  - *SortDetails* - Description of the sorting capabilities supported by the Retrieval Manager. Information provided includes *database* name, sort keys, key descriptions, key specification, key type and case sensitivity of the key. There are currently no sort requirements on the CIP and therefore *SortDetails* is not needed for the CIP.
  - *ProcessingInfo* - This provides descriptive information about how the *target* believes retrieved data must be processed by the client for presentation to the user. Various instructions can be provided per *database* and in more than one abstract syntax. An *Explain record* of this type is distinguished by *database*, processing context, name and object identifier. *ProcessingInfo* that can be provided includes *database* name, processing information context, processing information name, instructions OID, instruction descriptions and machine processable instructions. *ProcessingInfo* is not required to be provided for the CIP.

Note that the functionality of the CIP has been determined by the current baseline ICS URD requirements, which were derived independently of the Z39.50 facilities. Further CIP engineering and requirements analysis may indicate that the CIP would benefit from the non utilised Z39.50 facilities in which case they could of course be incorporated in CIP releases beyond Release B.

In Table 3-21 the ASN.1 description of the Explain records supported by the CIP is provided.

Table 3-37: Explain Record

ASN.1 Definition	Meaning

`Explain-Record ::= CHOICE{`
`  -- Each of these may be used as search term`
`  -- when Use attribute is 'explain-category'.`
`  targetInfo [0] IMPLICIT TargetInfo,`
`  databaseInfo [1] IMPLICIT DatabaseInfo,`
`  schemaInfo [2] IMPLICIT SchemaInfo,`
`  tagSetInfo [3] IMPLICIT TagSetInfo,`
`  recordSyntaxInfo [4] IMPLICIT RecordSyntaxInfo,`
`  attributeSetInfo [5] IMPLICIT AttributeSetInfo,`
`  termListInfo [6] IMPLICIT TermListInfo,`
`  extendedServicesInfo [7] IMPLICIT ExtendedServicesInfo,`
`  attributeDetails [8] IMPLICIT AttributeDetails,`
`  termListDetails [9] IMPLICIT TermListDetails,`
`  elementSetDetails [10] IMPLICIT ElementSetDetails,`
`  retrievalRecordDetails [11] IMPLICIT RetrievalRecordDetails,`
`  sortDetails [12] IMPLICIT SortDetails,`
`  processing [13] IMPLICIT ProcessingInformation,`
`  variants [14] IMPLICIT VariantSetInfo,`
`  units [15] IMPLICIT UnitInfo,`
`  categoryList [100] IMPLICIT CategoryList}`
	An **Explain-Record** can be of one of the following kind:
	• **targetInfo** provides information about a target. There will be only one such record per target.
	• For each collection a **databaseInfo** Explain record is defined. Also for the collection database itself a **databaseInfo** Explain record is available.
	• **schemaInfo** includes descriptive information about a database schema. There is one Explain record per schema supported.
	• **tagSetInfo** contains information about a given tag set.
	• **recordSyntaxInfo** provides descriptive information about a record syntax. There is one Explain record for each abstract record syntax supported by the target.
	• **attributeSetInfo** describes an attribute set. There is one record for each attribute set supported by the target.
	• Information about term-lists are contained in **termListInfo**.
	• **extendedServicesInfo** provides information about an extended service. For each of the extended services supported by the target one Explain record is defined.
	• **attributeDetails** contains information for each attribute. There is one Explain record for each supported database.
	• Descriptive information about a term-list is included in **termListDetails.** There is one record for each term-list listed by TermListInfo record.
	• **elementSetDetails** describes an element set. There is one Explain record for each element set for each record syntax for each database.
	• **retrievalRecordDetails** includes a descriptive information about the elements of a retrieval record. Note the elements are relative to a database schema. There is one such Explain record for each database for each schema for each record syntax.
	• **sortDetails** describes the sorting capabilities supported by a target.
	• **processing** provides instructions on how the target believes the data must be processed by the origin for presentation to the user.
	• In **variants** a descriptive information about a variant set definition supported by the target is provided.
	• **units** describes the unit system definitions supported by the target.
	• **categoryList** contains the list of the Explain categories supported by the target. There is one such record for a target.

3.5.9.2.1 Target Server Information

There are three types of information that apply across Retrieval Managers: ***TargetInfo***, ***ExtendedServices*** and ***CategoryList*** details.

TargetInfo

This provides information about the Retrieval Manager and there will be only one ***TargetInfo*** *Explain record* per Retrieval Manager. It will be used primarily to understand the Retrieval Manager's capacity for *result set* and search management.

Table 3-38: TargetInfo

ASN.1 Definition	Meaning

TargetInfo ::= SEQUENCE {
  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
  -- Key elements follow:
  name [1] IMPLICIT InternationalString,
  -- Non-key brief elements follow:
  recent-news [2] IMPLICIT HumanString OPTIONAL,
  icon [3] IMPLICIT IconObject OPTIONAL,
  namedResultSets [4] IMPLICIT BOOLEAN,
  multipleDBsearch [5] IMPLICIT BOOLEAN,
  maxResultSets [6] IMPLICIT INTEGER ~~OPTIONAL~~,
  maxResultSize [7] IMPLICIT INTEGER OPTIONAL,
  maxTerms [8] IMPLICIT INTEGER OPTIONAL,
  timeoutInterval [9] IMPLICIT IntUnit ~~OPTIONAL~~,
  welcomeMessage [10] IMPLICIT HumanString OPTIONAL,
  -- non-brief elements follow:
  -- 'description' esn retrieves the following two (as well as brief):
  contactInfo [11] IMPLICIT ContactInfo OPTIONAL,
  description [12] IMPLICIT HumanString OPTIONAL,
  nicknames [13] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
  usage-restrictions [14] IMPLICIT HumanString OPTIONAL,
  paymentAddr [15] IMPLICIT HumanString OPTIONAL,
  hours [16] IMPLICIT HumanString OPTIONAL,
  dbCombinations [17] IMPLICIT SEQUENCE OF DatabaseList OPTIONAL,
  addresses [18] IMPLICIT SEQUENCE OF NetworkAddress OPTIONAL,
  languages [101] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
  -- Languages supported for message strings. Each is a
  -- three language code from Z39.53-1994.
-- characterSets [102] this tag reserved for "character sets supported
-- for name and message strings".
  -- commonAccessInfo elements list objects the target supports. All objects
  -- listed in AccessInfo for any individual database should also be listed
  -- here.
  commonAccessInfo [19] IMPLICIT AccessInfo ~~OPTIONAL~~}
	The **TargetInfo** Explain record contains the following:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **name** provides the human readable name of the target. The **name** serves as the key to uniquely identify the Explain record.
	• **recent-news** includes texts which are of interest to the user community.
	• **icon** which is used to represent the target (in machine presentable form).
	• **namedResultSets** is a flag to indicate if support for named result sets is available at the target.
	• **multipleDBsearch** is a flag to indicate whether multiple databases can be searched
	• **maxResultSets** is the maximum number of concurrent result sets supported by the RM.
	• **maxResultSize** provides the maximum size (in records) of a result set.
	• **maxTerms** indicates the maximum number of terms allowed in one single search request.
	• **timeoutInterval** is the interval after which the target will close the connection if no activity has occurred.
	• **welcomeMessage** is a welcome message from the target to be displayed at the origin.
	• **contactInfo** provides contact information about the organisation supporting the target. The **ContactInfo** definition is given in Section 3.5.9.2.4.
	• **description** is a human readable text about the target.
	• **nicknames** provides alternate names for the target .
	• **usage-restrictions** defines the general restrictions pertaining the target .
	• **paymentAddr** gives the payment (business) address for the organisation running the target .

ASN.1 Definition	Meaning
	• **hours** indicates the hours of operations of the target . • **dbCombinations** gives the list of supported database combinations. This is not needed for the CIP, since the collection concept is introduced. • **addresses** provides the address, e.g. the port and Internet address, of the target . The definition of **NetworkAddress** is given in Section 3.5.9.2.4. • **languages** describes the languages the target supports for message strings (PDUs). For the CIP, English must be supported. • **characterSets** flags the character sets supported for name and message strings.

ExtendedServicesInfo

This provides information about the *extended services* supported by the *target*. Information includes the OID and name of the *extended service*, flags indicating details about the service, descriptive text and ASN.1 definitions.

The *extended services* supported by CIP are defined in Section 3.5.7 .

For each *Extended Service* supported there is one ***ExtendedServicesInfo*** *Explain record* available at the *target* (Retrieval Manager).

Table 3-39: ExtendedServicesInfo

ASN.1 Definition	Meaning

ExtendedServicesInfo ::= SEQUENCE {
commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
-- Key elements follow:
type [1] IMPLICIT OBJECT IDENTIFIER,
-- Non-key brief elements follow:
name [2] IMPLICIT InternationalString OPTIONAL,
-- should be supplied if privateType is 'true'
privateType [3] IMPLICIT BOOLEAN,
restrictionsApply [5] IMPLICIT BOOLEAN, -- if 'true' see 'description'
feeApply [6] IMPLICIT BOOLEAN, -- if 'true' see 'description'
available [7] IMPLICIT BOOLEAN,
retentionSupported [8] IMPLICIT BOOLEAN,
waitAction [9] IMPLICIT INTEGER{
waitSupported (1),
waitAlways (2),
waitNotSupported (3),
depends (4),
notSaying (5)},
-- non-brief elements follow:
-- To get brief plus 'description' use esn 'description'
description [10] IMPLICIT HumanString OPTIONAL,
-- to get above elements and 'specificExplain' use esn
--'specificExplain'
specificExplain [11] IMPLICIT EXTERNAL OPTIONAL,
-- Use oid of specific ES, and select choice
-- [3] 'explain'. Format to be developed in
-- conjunction with the specific ES definition.
-- to get all elements except 'specificExplain', use esn 'asn'
esASN [12] IMPLICITInternationalString OPTIONAL
-- the ASN.1 for this ES
}
	An **ExtendedServicesInfo** Explain record consists of:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **type**, which is the object identifier of the extended service.
	• a **name** under which the extended service is known.
	• **privateType** flag to indicate if this is a private extended service. This must be set for the CIP Order ES.
	• **restrictionsApply** flag to indicate whether restrictions apply for the use of the extended service.
	• **feeApply** flag to indicate whether fees applies for the use of the extended service.
	• **available** flag to indicate if the extended service is available.
	• **retention** flag to indicate if retention is supported by the extended service.
	• **waitAction** allows to designate the level of wait-action supported for an extended service. For CIP **waitSupported** or **waitAlways** are the only two alternatives.
	• **description** provides a human readable text to the client.
	• **specificExplain** includes explain elements specific to this extended services.
	• **esASN** is the ASN.1 module for the Explain definition.

CategoryList

This provides information about which ***Explain categories*** are supported by the *target*. There is one ***CategoryList*** *Explain record* per *target (*Retrieval Manager).

The primary use of the ***CategoryList*** is to allow clients to learn about *Explain categories* and any extensions supported by the *target*.

Table 3-40: CategoryList

ASN.1 Definition	Meaning

CategoryList ::= SEQUENCE {
  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
  -- Only one record expected per Explain database. All elements
  -- appear in brief presentation.
  categories [1] IMPLICIT SEQUENCE OF CategoryInfo }
	The **CategoryList** Explain record consists of:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• For each of the supported explain information categories of the Retrieval Manager a **CategoryInfo** element is provided in **categories**.

CategoryInfo ::= SEQUENCE {
  category [1] IMPLICIT InternationalString,
  originalCategory [2] IMPLICIT InternationalString OPTIONAL,
  description [3] IMPLICIT HumanString OPTIONAL,
  asn1Module [4] IMPLICIT InternationalString OPTIONAL}
	Each **CategoryInfo** is defined as:
	• **category** is the search term used in conjunction with the use attribute of Explain category to search for records of this category.
	• **originalCategory** is the original search term. (This is needed only for information categories where the Retrieval Manager is supporting a revision of the original definition of a category).
	• **description** of the category.
	• **asn1Module** is the ASN.1 definition of the record for this category.

3.5.9.2.2 CIP Data Information

Static information can be stored to enable the client to understand entities specific to the *target* Retrieval Manager, so allowing the client to use Retrieval Managers without having a priori knowledge of CIP related data definitions.

Attribute sets information, *tag sets* information, *schema information* and *record syntaxes* are particularly important *Explain* concepts for Z39.50 and the CIP. The information contained in Appendices A, B and C provide inputs for the definition of a CIP *Explain database*. There are six types of CIP data information: ***RecordSyntaxInfo***, ***SchemaInfo***, ***UnitInfo***, ***AttributeSetInfo*** and ***TagSetInfo.***

RecordSyntaxInfo

This provides descriptive information about a *record syntax*. A *record syntax* is an abstract syntax requested by the *origin* (client) or used by the *target* to represent *retrieval records*. In CIP the following *record syntax*es are supported: *Explain*, *SUTRS*, *GRS*-1 and *Extended Services*. For each of these *record syntaxes* there will be one ***RecordSyntaxInfo*** Explain record defined per Retrieval Manager.

The ***RecordSyntaxInfo*** is important for the client to be able to understand the format of the *retrieval records* (even if these formats are defined in this specification, it permits greater flexibility for the future.

Table 3-41: RecordSyntaxInfo

ASN.1 Definition	Meaning

RecordSyntaxInfo ::= SEQUENCE {
  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
  -- Key elements follow:
  recordSyntax [1] IMPLICIT OBJECT IDENTIFIER,
  -- Non-key brief elements follow:
  name [2] IMPLICIT InternationalString,
  -- non-brief elements follow:
  transferSyntaxes [3] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
  description [4] IMPLICIT HumanString OPTIONAL,
  asn1Module [5] IMPLICIT InternationalString OPTIONAL,
  abstractStructure [6] IMPLICIT SEQUENCE OF ElementInfo OPTIONAL
  -- Omitting abstractStructure only means target isn't using Explain to
  -- describe the structure, not that there is no structure.
}
	Each **RecordSyntaxInfo** Explain Record consists of:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **recordSyntax**, which is the object identifier of the abstract record syntax.
	• **name** by which the abstract record syntax is known.
	• **transferSyntaxes** includes all object identifiers of the transfer syntaxes that are supported for this abstract record syntax
	• **description** is a human readable textual description of the abstract record syntax.
	• **ans1 Module** is ASN.1 definition of the abstract record syntax.
	• **abstractStructure** is the record structure defined by this syntax.

AttributeSetInfo

An important requirement of the CIP is to provide data definition access and retrieval to the client for visibility to the end user. This is supported via the ***AttributeSetInfo*** *Explain data*. Inputs for the ***AttributeSetInfo*** *Explain data* is provided in Appendix A.

This provides descriptive information about a *target attribute set*, with one record per supported *attribute set*. An *attribute set* is a set of *attribute types*, and for each one of these types, a set of *attribute values*. Each type is represented by an integer, unique within that set and each value for a given type is unique within that type.

The definition of *attribute sets* via the *Explain database* is a crucial concept for the CIP. An *attribute set* is defined for each collection so that the searchable *attributes* for the collection and/or the products it contains are defined and can be accessed via the *Explain facility*, i.e. data can then be retrieved for use by the client and the Retrieval Manager itself, to aid data searching.

Note that not all of the information is mandatory for the CIP (e.g. ***description***), but for effective administration, the additional information could be provided.

Table 3-42: AttributeSetInfo

ASN.1 Definition	Meaning
AttributeSetInfo ::= SEQUENCE { commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~, -- Key elements follow: attributeSet [1] IMPLICIT AttributeSetId, -- non-key brief elements follow: name [2] IMPLICIT InternationalString, -- non-brief elements follow: attributes [3] IMPLICIT SEQUENCE OF AttributeType OPTIONAL, -- mandatory in full record description [4] IMPLICIT HumanString OPTIONAL}	Each **AttributeSetInfo** consists of: • **commonInfo** holds general information (see details in Section 3.5.9.2.4.) • **attributeSet** which defines the object identifier for the specific attribute set. • **name** of the attribute set. • for each **AttributeType** (e.g. Use , Relation) existing in the attribute set an entry is provided in **attributes** . • **description** in human readable form of the attribute set. Appendix A. provides input for the elements in **AttributeSetInfo**.
AttributeType ::= SEQUENCE { name [0] IMPLICIT InternationalString OPTIONAL, description [1] IMPLICIT HumanString OPTIONAL, attributeType [2] IMPLICIT INTEGER, attributeValues [3] IMPLICIT SEQUENCE OF AttributeDescription}	For each attribute type available in the attribute set an **AttributeType** entry is provided. The **AttributeType** entry is a combination of: • the **name** of the attribute type (e.g. Use , Relation , Truncation, Structure) • a human readable **description** of the attribute type, describing how the specific attribute type is used. • an integer value for the **attributeType** (e.g. Use attribute has integer value "1", Relation attribute has integer value "2"). • **attributeValues** contains a list of attributes descriptions of that specific type.

AttributeDescription ::= SEQUENCE {
name [0] IMPLICIT InternationalString OPTIONAL,
description [1] IMPLICIT HumanString OPTIONAL,
attributeValue [2] StringOrNumeric,
equivalentAttributes [3] IMPLICIT SEQUENCE OF StringOrNumeric
OPTIONAL
-- each is an occurrence of'attributeValue' from AttributeDescription
-- for a different attribute. Equivalences listed here should be
-- derived from the attribute set definition, not from a particular
-- server's behavior.
}
	AttributeDescription contains:
	• the **name** of an attribute of a particular attribute type (e.g. the name of an attribute of attribute type "Structure" would be "Word").
	• a **description** of an attribute of a particular attribute type
	• **attributeValue** contains the value of the attribute
	• **equivalentAttributes** list the names of equivalent attributes

UnitInfo

This provides information about unit system definitions supported by the Retrieval Manager. Numeric terms can then be specified by the client and the CIP can use information provided in a ***UnitInfo*** *Explain record* to interpret the search and results without needing to understand it.

An example of a unit type could be *length* and a value *metres*.

The ***UnitInfo*** is an example of the power of the *Explain database*, where data retrieved from the *Explain database* can help the CIP user to understand specifics (e.g. via human understandable explanatory texts) of the *target* data which can vary from server to server, without the client needing to understand the differences.

Table 3-43: UnitInfo

ASN.1 Definition	Meaning
UnitInfo ::= SEQUENCE { commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~, -- Key elements follow: unitSystem [1] IMPLICIT InternationalString, -- No non-key brief elements -- Non-brief elements follow: description [2] IMPLICIT HumanString OPTIONAL, units [3] IMPLICIT SEQUENCE OF UnitType OPTIONAL } -- mandatory in full record	A **UnitInfo** Explain record includes: • **commonInfo** holds general information (see details in Section 3.5.9.2.4.) • the **name** of the unit system . In CIP the ISO 2955 unit system must be supported in order to be compatible with the CCSDS DEDSL standard[DEDSL]). • a **description** of the unit system . • **units** includes the list of all **UnitType** supported by the unit system
UnitType ::= SEQUENCE { name [0] IMPLICIT InternationalString OPTIONAL, description [1] IMPLICIT HumanString OPTIONAL, unitType [2] StringOrNumeric, units [3] IMPLICIT SEQUENCE OF Units}	Each **UnitType** consists of the following: • **name** of the unit type, e.g. ”Acceleration”, Energy”, “Frequency”, “Length”, “Mass”, “Time”, “Velocity”, “Volume”. • a **description** of the unit type • the **unitType** • **units** includes the list of all **Units**
Units ::= SEQUENCE { name [0] IMPLICIT InternationalString OPTIONAL, description [1] IMPLICIT HumanString OPTIONAL, unit [2] StringOrNumeric}	For each of the **Units** the following information is included: • **name** of the unit (e.g. for unit type “Length” a name of a unit might be “meter”). • unit **description** • **unit** includes the value of the unit

TagSetInfo

This provides information about *tag sets*. There is one such *Explain record* per *tag set*. A *tag set* provides *tag values* and recommended data types for a set of *elements*. Inputs for the ***TagSetInfo*** data is provided in Appendix B.

Each *element* in a *database record* is tagged and the *tag* acts as an identifier for the *element*. The numeric *tag* is understood both by the client and the Retrieval Manager.

Table 3-44: TagSetInfo

ASN.1 Definition	Meaning

TagSetInfo ::= SEQUENCE {
commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
-- Key elements follow:
tagSet [1] IMPLICIT OBJECT IDENTIFIER,
-- non-key brief elements follow:
name [2] IMPLICIT InternationalString,
-- non-brief elements follow:
description [3] IMPLICIT HumanString OPTIONAL,
elements [4] IMPLICIT SEQUENCE OF SEQUENCE {
elementname [1] IMPLICIT InternationalString,
nicknames [2] IMPLICIT SEQUENCE OF
InternationalString OPTIONAL,
elementTag [3] StringOrNumeric,
description [4] IMPLICIT HumanString OPTIONAL,
dataType [5] PrimitiveDataType OPTIONAL,
-- If the data type is expected to be structured,
-- that is described in the schema info, and
-- datatype is omitted here.
otherTagInfo OtherInformation ~~OPTIONAL~~}
OPTIONAL}
	A **TagSetInfo** Explain record includes:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **tagSet** which specifies the tag set object identifier
	• the **name** of the tag set.
	• a human readable textual **description** of the tag set
	• **elements** provides a list of all elements included in the tag set. For each element the following information is included:
	  • **elementName** is the name of the element. This is the 'Name' semantic attribute.
	  • **nicknames** define alternate names for the element. This is the 'Alias' semantic attribute.
	  • **elementTag** specifies the tag assigned to the element (i.e. identification of the element).
	  • **description** of the element. This is the 'Description' semantic attribute.
	  • **dataType** of the element (e.g. "numeric", "octetstring"). A definition of **PrimitiveDataType** is given in Section 3.5.9.2.4.
	  • **otherTagInfo** allows the specification of additional information about an element. This is used by the CIP to include the semantic attributes which are not covered by the Z39.50 standard using the 'AdditionalSemanticAttributes' EXTERNAL (see Appendix E).

SchemaInfo

A *database schema* enables common understanding to be shared by the client and server of information contained in *database records* at the server. This allows subsequent selection of portions of that information via an *element specification*. A *schema* defines an *abstract record structure*, which when applied to a *database record*, results in an *abstract database record*. Inputs for the ***SchemaInfo*** is provided in Appendix C.

The *schema* defines the *tags* used in the *database records*. Numeric *tags* can be selected from the CIP *tag sets* or they may be defined for a given *database* or set of *databases*. There is one *Explain record* for each schema supported by the *target*.

The CIP provides *schema* information indirectly with use of the *tag set* information, as follows: For the *tag path*, information required is ***tagType***, ***tagValue*** and appropriate information about the data type (e.g. structured or primitive where examples of possible primitive types include ***octetString***, ***numeric***, ***date***, ***string***, ***OID***, ***intUnit***, etc.).

Table 3-45: SchemaInfo

ASN.1 Definition	Meaning

SchemaInfo ::= SEQUENCE {
commonInfo [0] IMPLICIT CommonInfo OPTIONAL,
-- Key elements follow:
schema [1] IMPLICIT OBJECT IDENTIFIER,
-- Non-key brief elements follow:
name [2] IMPLICIT InternationalString,
-- Non-brief elements follow:
description [3] IMPLICIT HumanString OPTIONAL,
tagTypeMapping [4] IMPLICIT SEQUENCE OF SEQUENCE {
tagType [0] IMPLICIT INTEGER,
tagSet [1] IMPLICIT OBJECT IDENTIFIER ~~OPTIONAL~~,
-- If tagSet is omitted, then this tagType is for a
-- tagSet locally defined within the schema that
-- cannot be referenced by another schema.
defaultTagType [2] IMPLICIT NULL OPTIONAL} ~~OPTIONAL~~,
recordStructure [5] IMPLICIT SEQUENCE OF ElementInfo ~~OPTIONAL~~}
	A **SchemaInfo** Explain record consists of:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **schema** specifies the object identifier, which is used to uniquely identify the schema.
	• the **name** of the schema.
	• a human readable textual **description** of the schema.
	• **tagTypeMapping** gives the mapping between the **tagSet** and the **tagType**. Moreover, it indicates if the **tagSet** identified by the **tagType** is considered as the **defaultTagType.** Note that the CIP tagSet **{ANSI-standard-Z39.50 14 2000 99 1}** is identified by **tagType** 4 and is the **defaultTagType**.
	• the abstract **recordStructure** defined by the schema. The details of the **ElementInfo** are provided in Section 3.5.9.2.4.

3.5.9.2.3 Target Database Information

The majority of the dynamic content of an *Explain database* relates to a *database* or set of *databases* (i.e. collections) and the associated services (i.e. a catalogue) on a *target* server (Retrieval Manager). Each *database* is described by a ***DatabaseInfo*** record and additional records describe specific aspects of the *database*. This includes query details, term (index) lists, sorting details, processing information etc. By accessing and retrieving such data, an end user can form more appropriate search queries and increase the success and accuracy of search results.

DatabaseInfo

This provides detailed descriptions of Retrieval Manager *target databases*, *database* related restrictions and parameters.

The key *databases* form pertinent to the CIP domain are *databases* of archive catalogue information e.g. a collection of product descriptors that describe data that can be retrieved from archives (i.e. a archive collection). In the CIP domain all collections (terminal and non-terminal) are considered as *databases*. There is one ***DatabaseInfo*** *Explain record* for each *database* supported. This *record* is located in the *Explain database* of the target in which the collection is defined, i.e. in the local *Explain database* for local collections and in the corresponding remote *Explain database* for a remote collection. On initialisation of a *Z-association*, relevant *database* information can be retrieved from the *Explain database* and made available to the client and end user to confirm *database* information at the *target*.

Table 3-46: DatabaseInfo

ASN.1 Definition	Meaning

`DatabaseInfo ::= SEQUENCE {`
`  -- A target may provide "virtual databases" that are combinations of`
`  -- individual database. These databases are indicated by the`
`  -- presence of subDbs in the combination database's`
`  -- DatabaseDescription.`
`  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,`
`  -- Key elements follow:`
`  name [1] IMPLICIT DatabaseName,`
`  -- Non-key brief elements follow:`
`  explainDatabase [2] IMPLICIT NULL OPTIONAL,`
`  -- If present, this database is the Explain`
`  -- database, or an Explain database for a`
`  -- different server, possibly on a`
`  -- different host. The means by which that`
`  -- server may be accessed is not addressed`
`  -- by this standard. One suggested`
`  -- possibility is an implementor agreement`
`  -- whereby the database name is a url which`
`  -- may be used to connect to the server.`
`  nicknames [3] IMPLICIT SEQUENCE OF DatabaseName OPTIONAL,`
`  icon [4] IMPLICIT IconObject OPTIONAL,`
`  user-fee [5] IMPLICIT BOOLEAN,`
`  available [6] IMPLICIT BOOLEAN,`
`  titleString [7] IMPLICIT HumanString OPTIONAL,`
`  -- Non-brief elements follow:`
`  keywords [8] IMPLICIT SEQUENCE OF HumanString OPTIONAL,`
`  description [9] IMPLICIT HumanString OPTIONAL,`
`  associatedDbs [10] IMPLICIT DatabaseList OPTIONAL,`
`  -- databases that may be searched in`
`  -- combination with this one`
`  subDbs [11 IMPLICIT DatabaseList OPTIONAL,`
`  -- When present, this database is a`
`  -- composite representing the combined`
`  -- databases 'subDbs'. The individual subDbs`
`  -- are also available.`
`  disclaimers [12] IMPLICIT HumanString OPTIONAL,`
`  news [13] IMPLICIT HumanString OPTIONAL,`
`  recordCount [14] CHOICE {`
`  actualNumber [0] IMPLICIT INTEGER,`
`  approxNumber [1] IMPLICIT INTEGER}`
`  OPTIONAL,`
`  defaultOrder [15] IMPLICIT HumanString OPTIONAL,`
`  avRecordSize [16] IMPLICIT INTEGER OPTIONAL,`
`  maxRecordSize [17] IMPLICIT INTEGER OPTIONAL,`
`  hours [18] IMPLICIT HumanString OPTIONAL,`
`  bestTime [19] IMPLICIT HumanString OPTIONAL,`
`  lastUpdate [20] IMPLICIT GeneralizedTime OPTIONAL,`
`  updateInterval [21] IMPLICIT IntUnit OPTIONAL,`
	A **DatabaseInfo** Explain record includes:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• Full database **name** (only one). When **DatabaseInfo** describes a collection, **name** contains the collection descriptor identifier (i.e. the value of the ItemDescriptorId schema element).
	The **name** of 'pseudo' default databases starts with '**DEF-DB**'. The default database that encompasses the complete CIP requirements has the reserved **name**: '**DEF-DB-CIP**'.(More detailed explanation is available under **TermListInfo** and **AttributeDetails** Explain records).
	• **explainDatabase** states whether the database is a Explain database, possibly of a different server. (Note that this will always be MISSING as the Explain database itself will not be in the Explain).
	• **nicknames** provides a list of short alternate names for a database
	• **icon** allows the specification of an icon object for representation of the database (in machine presentable form).
	• **user-fee** indicates whether costs are associated with the access to this database.
	• **available** indicates whether the database is available for access.
	• **titleString** includes a human readable name or title for the database (as opposed to the database name, which is meant to be short and not language dependent). When **DatabaseInfo** describes a collection, **titleString** contains the collection descriptor name (i.e. the value of the ItemDescriptorName schema element).
	• **keywords** provide an indication of whether the database also has a special purpose, in addition to being a target database, such as being a 'key access point' or 'root collection node'. The value for keywords are:
	  • **KEY**: a key access point.
	  • **ROOT**: for the root collection node of the CIP domain within the *target* Retrieval Manager.
	The **keywords** field is also used to state if a database (collection) uses a default or if it has its own **TermListInfo** Explain record. If a default is used the value "**TermListInfo = <*Name of Default Database*>"** is provided in the keywords field. Note that for **<*Name of Default Database*** > the name of the default database must be inserted. More detailed description is provided under the **TermListInfo** explain record.
	An analogue use of the **keywords** field is made in the case that default options

ASN.1 Definition	Meaning

`coverage [22] IMPLICIT HumanString OPTIONAL,`
`proprietary [23] IMPLICIT BOOLEAN OPTIONAL,`
`-- mandatory in full record`
`copyrightText [24] IMPLICIT HumanString OPTIONAL,`
`copyrightNotice [25] IMPLICIT HumanString OPTIONAL,`
`producerContactInfo [26] IMPLICIT ContactInfo OPTIONAL,`
`supplierContactInfo [27] IMPLICIT ContactInfo OPTIONAL,`
`submissionContactInfo [28] IMPLICIT ContactInfo OPTIONAL,`
`-- accessInfo lists items connected with the database. All listed items`
`-- should be in the target's AccessInfo.`
`accessInfo [29] IMPLICIT AccessInfo ~~OPTIONAL~~}`
	are used for the provision of **AttributeDetails**. If a default is used the value "**AttributeDetails = <*Name of Default Database*>"** is provided in the keywords field. A more detailed explanation is provided under **AttributeDetails** explain record.
	The **keywords** field may optionally include other values that are significant to the local catalogue system, which would be ignored by a generic CIP client/sever.
	• human readable **description** of the database
	• **associatedDbs** indicates the databases that the target allows (and possibly encourages) to be searched in combination with the database described.
	• **subDbs** is used when the **DatabaseInfo** describes a collection. **subDbs** allows to simplify the origin's configuration with regards to the collection hierarchy. **SubDbs** is empty if the collection is a terminal collection. Otherwise, **SubDbs** contains the list of the collection descriptor identifiers at the next lower level in the collection hierarchy (i.e. a reference to the **name** of the **DatabaseInfo** record for the collections at the next lower hierarchical level).
	• any **disclaimers** concerning this database.
	• **news** about the database are not needed since all collections have the last date of modification/creation associated with them.
	• **recordCount** provides an exact (**actualNumber**) and approximate (**approxNumber**) record count for the database (collection).
	• **defaultOrder** specifies the default order in which the records are returned.
	• **avRecordSize** is an estimate of the average record size in bytes
	• **maxRecordSize** specifies the maximum record size in bytes.
	• **hours** of operations is not needed for the database. Only for the Retrieval Manager this information is provided (see TargetInfo).
	• **bestTime** of access to the database.
	• **Update**, **updateInterval**, **coverage**, **proprietary**, **copyrightText**, **copyrightNotice**, **producerContactInfo**, **supplierContactLastnfo**, **submissionContactInfo** is not needed for the CIP, since it is already provided in the Collection descriptor.
	• **accessInfo** provides details about the query types, schemas, attribute sets, record syntaxes, etc. related to the database. Details can be found in Section 3.5.9.2.4. This feature is mandatory since **AccessInfo** provides information about the schemas, attribute sets, etc. which is needed to dynamically configure a client.

AttributeDetails

This provides information for each *attribute*. According to Z39.50 there should be one ***AttributeDetails*** *Explain record* for each supported *database* (collection).

For CIP, this principle has been slightly altered, since frequently *databases* (collections) held within one Retrieval Manager and even across Retrieval Managers use the same subset of CIP (or local) attribute combinations/attribute sets. In order to avoid unnecessary duplication of information, ***AttributeDetails*** *Explain records* (for 'pseudo' default *databases*) can be established. These defaults lists can then be referenced by all those *databases* (collections) to which the default option applies. The following procedure must be followed to create and reference default ***AttributeDetails*** *Explain records*:

- a 'pseudo' default *database* must be first created. The name of any default *database* must start with '**DEF-DB**'. The special default database that covers all CIP requirements is given the reserved name: '**DEF-DB-CIP**'.
- a *DatabaseInfo Explain record* (using the previously defined name) is created for the default *database.*
- a default ***AttributeDetails*** *Explain record* that encompasses all the desired options is then created. The **databaseName** that is included in the default ***AttributeDetails*** *Explain record* refers to the default *database*.
- when another *database* (collection) wants to use this default ***AttributeDetails*** *Explain record*, it needs to insert in the keyword field of its *DatabaseInfo Explain record* the following statement: "**AttributeDetails = < *Name of Default Database*>,** where **< *Name of Default Database*>** must state the name of the default *database* (e.g. **'DEF-DB-CIP'**).
- for any *database* (collection) that refers to a default ***AttributeDetails*** *Explain record* via the keyword field in the its *DatabaseInfo Explain record* , no own ***AttributeDetails*** *Explain record* needs to be defined.

Inputs for the ***AttributeDetails*** is provided in Appendix A.

Table 3-47: AttributeDetails

ASN.1 Definition	Meaning
AttributeDetails ::= SEQUENCE { commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~, -- Key elements follow: databaseName [1] IMPLICIT DatabaseName, -- Non-brief elements follow: attributesBySet [2] IMPLICIT SEQUENCE OF AttributeSetDetails OPTIONAL, -- mandatory in full record attributeCombinations [3] IMPLICIT AttributeCombinations OPTIONAL}	An **AttributeDetails** Explain record contains: • **commonInfo** holds general information (see details in Section 3.5.9.2.4.) • **databaseName** indicates the name of the database to which this attribute information applies. In the case that default attribute details are created, the name of a 'pseudo' database is inserted here. The **databaseName** of such 'pseudo' (default) databases must start with '**DEF-DB**'. A 'pseudo' database that supports the complete CIP requirements must have the **databaseName**: '**DEF-DB-CIP'**. • **attributesBySet** defines for each attribute set supported by the database their **AttributeSetDetails**. • **attributeCombinations** which lists all attributes combinations supported for the database.
AttributeSetDetails ::= SEQUENCE { attributeSet [0] IMPLICIT AttributeSetId, attributesByType [1] IMPLICIT SEQUENCE OF AttributeTypeDetails}	**AttributeSetDetails** provides for each attribute set supported by the database: • **attributeSet** which is the object identifier of the attribute set. • **attributesByType** which describes each single attribute type supported in the attribute set.
AttributeTypeDetails ::= SEQUENCE { attributeType [0] IMPLICIT INTEGER, defaultIfOmitted [1] IMPLICIT OmittedAttributeInterpretation OPTIONAL, attributeValues [2] IMPLICIT SEQUENCE OF AttributeValue OPTIONAL } -- If no attributeValues are supplied, all values of this type are -- fully supported, and the descriptions in AttributeSetInfo are adequate	For each attribute within the set **AttributeTypeDetails** includes: • **attributeType** which provides the type of the attribute, e.g. 'Use attribute'. • **defaultIfOmitted** which specifies the default value that is applied if the attribute is omitted and which provides a description of the default behaviour. • **attributeValues** which provides details on the attribute value.
OmittedAttributeInterpretation ::= SEQUENCE { defaultValue [0] StringOrNumeric OPTIONAL, -- A default value is listed if that's how -- the server works defaultDescription [1] IMPLICIT HumanString OPTIONAL } -- The human-readable description should -- generally be provided. It is legal for -- both default elements to be missing, -- which means that the target will allow -- the attribute type to be omitted, but -- isn't saying what it will do.	The **OmittedAttributeInterpretation** provides: • **defaultValue** which specifies the default value the Retrieval Manager will use in the case that the attribute is omitted. • **defaultDescription** which provides a human readable description of the default behaviour.

ASN.1 Definition	Meaning

`AttributeValue ::= SEQUENCE {`
`  value  [0] StringOrNumeric,`
`  description  [1] IMPLICIT HumanString OPTIONAL,`
`  subAttributes  [2] IMPLICIT SEQUENCE OF StringOrNumeric OPTIONAL,`
`  superAttributes  [3] IMPLICIT SEQUENCE OF StringOrNumeric OPTIONAL,`
`  partialSupport  [4] IMPLICIT NULL OPTIONAL`
`  -- partialSupport indicates that an attributeValue is accepted, but may not`
`  -- be processed in the"expected" way. One important reason for this is`
`  -- composite databases: in this case partialSupport may indicate that only`
`  -- some of the subDbs support the attribute, and others ignore it.`
`  }`
	In **AttributeValue** the following elements are provided for attribute value:
	• **value** gives the attribute value.
	• a human readable text **description** of the attribute.
	• **subAttributes** (applies for Use attributes) lists alternative values (different Use attributes) that allow access to the same aspect of the record, but in a greater detail.
	• **superAttributes** (applies for Use attributes) lists alternative values that allow access to the same aspect of the record, but in a coarser detail.
	• **partialSupport** states whether the value is only 'partially supported'. This is mandatory for local attributes.

TermListInfo

This provides information about the lists of valids that are available for a specific *database* (collection). According to Z39.50[Z3950] there should be one ***TermListInfo*** *Explain record* for each supported *database* (collection).

For CIP, this principle has been slightly altered, since frequently *databases* (collections) held within one Retrieval Manager and even across Retrieval Managers use the same set of CIP or local lists of valids. In order to avoid unnecessary duplication of information, default lists of valids, that means default ***TermListInfo*** *Explain records* (for 'pseudo' default *databases*) can be established. These defaults lists of valids can then be referenced by all those *databases* (collections) to which the default option applies. The following procedure must be followed to create and reference default ***TermListInfo*** *Explain records*:

- a 'pseudo' default *database* must be first created. The name of any default *database* must start with '**DEF-DB**'. The special default database that covers all CIP requirements is given the reserved name: '**DEF-DB-CIP**'.
- a *DatabaseInfo Explain record* (using the previously defined name) is created for the default *database.*
- a default ***TermListInfo*** *Explain record* that encompasses all the desired lists of valids is then created. The **databaseName** that is included in the default ***TermListInfo*** *Explain record* refers to the default *database*.
- when another *database* (collection) wants to use this default ***TermListInfo*** *Explain record*, it needs to insert in the keyword field of its *DatabaseInfo Explain record* the following statement: "**TermListInfo = < *Name of Default Database*>,** where **< *Name of Default Database*>** must state the name of the default *database* (e.g. **'DEF-DB-CIP'**).
- for any *database* (collection) that refers to a default ***TermListInfo*** *Explain record* via the keyword field in the its *DatabaseInfo Explain record* , no own ***TermListInfo*** *Explain record* needs to be defined.

Note that for each term list that is referred to in **termLists** an associated ***TermListDetails*** *Explain record* must exist.

The CIP lists of valids, input to ***TermListDetails,*** are defined in [VALID]. Note that lists of valids that are not yet available in [VALID] are inserted into the Explain database in the same way as the CIP ones.

Table 3-48: TermListInfo

ASN.1 Definition	Meaning

TermListInfo ::= SEQUENCE{
commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
-- Key elements follow:
databaseName [1] IMPLICIT DatabaseName,
-- Non-key brief elements follow:
termLists [2] IMPLICIT SEQUENCE OF SEQUENCE{
name [1] IMPLICIT InternationalString,
title [2] IMPLICIT HumanString OPTIONAL,
-- Title is for users to see and can
-- differ by language.
-- Name, on the other hand, is typically
-- a short string not necessarily meant
-- to be human- readable, and not
-- variable by language.
searchCost [3] IMPLICIT INTEGER {
optimized (0),
-- The attribute (or combination)
-- associated with this list will
-- do fast searches.
normal (1),
-- The attribute (combination) will
-- work expected. So there's
-- probably an index for the
-- attribute (combination) or some
-- similar mechanism.
expensive (2),
-- Can use the attribute
-- (combination), but it might not
-- provide satisfactory results.
-- Probably there is no index, or
-- post-processing of records is
-- required.
filter (3)
-- can't search with this
-- attribute (combination) alone.
} OPTIONAL,
scanable [4] IMPLICIT BOOLEAN,
-- 'true' means this list can be
-- scanned.
	An **TermListInfo** Explain record contains:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **databaseName** indicates the name of the database (collection) to which this term list information applies. In the case that a default term list information is created, the name of a 'pseudo' database is inserted here. The databaseName of such 'pseudo' (default) databases must start with 'DEF-DB'. A 'pseudo' database that supports the complete CIP requirements must have the **databaseName**: '**DEF-DB-CIP'**.
	• **termLists** provides information on all the term lists that are available for the database (collection):
	  • **name** provides the name of a term list that is supported by the database. Details about the list are defined in the associated **TermListDetails** Explain record. A **name** maps always into a **termListName** in the **TermListDetails** Explain record. Note that in CIP in the most cases (exception are hierarchical valids) the **name** maps directly to the name of a use attribute (either CIP or locally defined).
	  • **title** provides the title of the list (possibly in different languages).
	  • **searchCost** gives indications about the performance of searches using the attribute associated with this list.
	  • **scanable** states whether this list can be scanned. The value 'true' indicates that it can be scanned.

ASN.1 Definition	Meaning
broader [5] IMPLICIT SEQUENCE OF InternationalString OPTIONAL, narrower [6] IMPLICIT SEQUENCE OF InternationalString OPTIONAL -- broader and narrower list -- alternative term lists related to -- this one. The term lists so listed -- should also be in this termLists -- structure. } -- no non-brief elements }	• **broader** indicates an alternate 'broader; list that is related to this one. • **narrower** indicates an alternate 'narrower' list that is related to this one.

TermListDetails

For each term list (list of valids) that is available, a ***TermListDetails*** *Explain record* is created. Note that this applies equally to CIP defined and locally defined lists of valids. The CIP lists of valids are defined in [VALID].

In order to support an hierarchical approach to keywords, where the appearance of a term list is dependent on the valid (sampleTerm) of a parent term list, the following procedures is defined:

- for the 'child' term list a statement is inserted in the **description** in the ***TermListDetails*** *Explain record* that states the condition of the 'parent' term list (the valid of the 'parent' term list) that triggers the appearance of the 'child' term list.
- the statement must use the following convention:
  - **(WHERE <*Name of parent term list* > = <*valid*>)'**
  - **<*Name of parent term list*>** provides the name of the parent term list and
  - **<*valid*>** refers to the sampleTerm of the parent term list that triggers the appearance of the current term list
  - for example in [VALID]: "Aerosols, Air Quality, Altitude, Atmospheric Chemistry, …." only appear under the condition **(WHERE TopicKeyword = ATMOSPHERE)**).

Table 3-49: TermListDetails

ASN.1 Definition	Meaning

`TermListDetails ::= SEQUENCE{ -- one for each termList in TermListInfo`
`commonInfo [0] IMPLICIT CommonInfo OPTIONAL,`
`-- Key elements follow:`
`termListName [1] IMPLICIT InternationalString,`
`-- Non-key elements (all non-brief) follow:`
`description [2] IMPLICIT HumanString OPTIONAL,`
`attributes [3] IMPLICIT AttributeCombinations OPTIONAL,`
`-- Pattern for attributes that hit this list.`
`-- Mandatory in full record`
`scanInfo [4] IMPLICIT SEQUENCE {`
`maxStepSize [0] IMPLICIT INTEGER OPTIONAL,`
`collatingSequence [1] IMPLICIT HumanString OPTIONAL,`
`increasing [2] IMPLICIT BOOLEAN OPTIONAL}`
`OPTIONAL,`
`-- Occurs only if list is scanable.`
`-- If list is scanable and if scanInfo is omitted,`
`-- target doesn't consider these important.`
`estNumberTerms [5] IMPLICIT INTEGER OPTIONAL,`
`sampleTerms [6] IMPLICIT SEQUENCE OF Term OPTIONAL}`
	An **TermListDetails** Explain record contains:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.).
	• **termListName** provides the name of the term list. Note that in CIP in the most cases (exception are hierarchical valids) the **termListName** maps directly to the name of a use attribute (either CIP or locally defined).
	• **description** is normally used to provide free-format descriptive text about the term list. For hierarchical term lists, **description** is used in order to state under which condition of its parent list the term list appears (for more details see description above table)
	• **attributes** maps a term list to a single use attribute in one combination. (Note that in theory a term list could be mapped to a particular combination of use and relation attributes. Or a single term list could be mapped to many single use attribute combinations. But in the CIP, only the mapping of one term list to a single use attribute is used).
	• **scanInfo** provides information related to the scanning of the list.
	  • **maxStepSize** states the maximum step size that is supported.
	  • **collatingSequence** describes the collating sequence (e.g. ASCII, EBCDIC) in human readable text.
	  • **increasing** describes in which order (descending or ascending) is used.
	• **estNumberTerms** lists the estimated number of terms in the list.
	• **sampleTerms** lists all the valids that are included in the list.

RetrievalRecordDetails

This provides descriptive information about the *elements* of a *retrieval record* (*elements* are relative to a *database schema*). There is one such *Explain record* for each *database* for each *record syntax*. Inputs for the ***RetrievalRecordDetails*** are provided in Appendices B and C.

For retrieval of collection information, CIP *element sets* will be defined in the *Explain database*. A 'Full' set and a 'Brief' set are defined as defaults. The Full set will be the default for a full retrieval and the 'Brief' set will be the default for a brief retrieval. A 'CIP Full' set, which contains all the CIP elements, will also be defined. A 'Summary' set is defined for retrieving elements relevant for interoperability with Z39.50 Version-2 profiles. There is also a 'Browse' set for retrieving only browse data and a 'Local Attributes' set for the retrieval of the definition and content of the local attributes. Also a 'Collection Member' set for retrieving collection tree hierarchy information and a 'Options' set for retrieving the service options is provided. Further local *retrieval records* can then be defined in the *Explain database* as required by the site administrators to cover local *database* fields.

It is important to note that *retrieval record* specifications can be different to the *attribute sets*. Although *attributes* are effectively the primary *elements* of a *retrieval record* specification, additional *elements* associated may also be added to the *retrieval record*.

Table 3-50: RetrievalRecordDetails

ASN.1 Definition	Meaning

RetrievalRecordDetails ::= SEQUENCE {
  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,
  -- Key elements follow:
  databaseName [1] IMPLICIT DatabaseName,
  schema [2] IMPLICIT OBJECT IDENTIFIER,
  recordSyntax [3] IMPLICIT OBJECT IDENTIFIER,
  -- Non-brief elements follow:
  description [4] IMPLICIT HumanString OPTIONAL,
  detailsPerElement [5] IMPLICIT SEQUENCE OF PerElementDetails OPTIONAL
  -- mandatory in full record
}
	A **RetrievalRecordDetails** Explain record includes:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• the **databaseName**, **schema** and **recordSyntax** to which the Explain record pertains.
	• a human readable **description** of the retrieval record
	• **detailsPerElement** describes for each element contained in the syntax its specific details. **PerElementDetails** can be found in Section 3.5.9.2.4.

ElementSetDetails

This provides detailed descriptive information about *element sets*, with one *Explain record* per supported *element set* for each *record syntax* for each *database*. *ElementSetDetails* describes the way that *database records* are mapped to *record elements*. *Element set* information enables the client to identify the sets of retrieval formats that are available. For the CIP, there will be Full, CIP Full, Brief, Summary, Browse, Local Attributes, Collection Member and Options (see Section 3.5.3.3.2 and Appendix C.4).

Element set fields will map onto *elements* defined in ***SchemaInfo***. There may be additional information in the *element set* for the retrieval such as date, report information, etc.

Table 3-51: ElementSetDetails

ASN.1 Definition	Meaning

`ElementSetDetails ::= SEQUENCE {`
`  -- ElementSetDetails describes the way that database records are mapped to`
`  -- record elements. This mapping may be different for each combination of`
`  -- database name and element set. The database record description is a`
`  -- schema, which may be private to the target. The schema's abstract`
`  -- record structure and tag sets provide the vocabulary for discussing`
`  -- record content; their presence in the Explain database does not imply`
`  -- support for complex retrieval specification.`
`  commonInfo [0] IMPLICIT CommonInfo ~~OPTIONAL~~,`
`  -- Key elements follow:`
`  databaseName [1] IMPLICIT DatabaseName,`
`  elementSetName [2] IMPLICIT ElementSetName,`
`  recordSyntax [3] IMPLICIT OBJECT IDENTIFIER,`
`  -- Non-key Brief elements follow:`
`  schema [4] IMPLICIT OBJECT IDENTIFIER,`
`  -- Non-brief elements follow:`
`  description [5] IMPLICIT HumanString OPTIONAL,`
`  detailsPerElement [6] IMPLICIT SEQUENCE OF PerElementDetails OPTIONAL`
`  -- mandatory in full record`
`  }`
	The **ElementSetDetails** Explain record includes:
	• **commonInfo** holds general information (see details in Section 3.5.9.2.4.)
	• **databaseName** is the name of the relevant database
	• **elementSetName** is the name of the element set.
	• the object identifier of the **recordSyntax** to which this Explain record pertains.
	• the object identifier of the **schema** for this element set.
	• **description** of the element set
	• **detailsPerElement** describes for each element contained in the syntax its specific details. **PerElementDetails** can be found in Section 3.5.9.2.4.

3.5.9.2.4 Auxiliary Definitions

This table includes all definitions which are used in various Explain categories, e.g. all *Explain category records*, have ***CommonInfo*** records which are that are used to hold general information about the particular category.

Table 3-52: Auxiliary Definitions

ASN.1 Definition	Meaning

`CommonInfo ::= SEQUENCE {`
`  dateAdded [0] IMPLICIT GeneralizedTime ~~OPTIONAL~~,`
`  dateChanged [1] IMPLICIT GeneralizedTime ~~OPTIONAL~~,`
`  expiry [2] IMPLICIT GeneralizedTime ~~OPTIONAL~~,`
`  humanString-Language [3] IMPLICIT LanguageCode OPTIONAL,`
`  -- following not to occur for brief:`
`  otherInfo OtherInformation OPTIONAL}`
	CommonInfo is used in all Explain records. It holds the following information:
	• **dateAdded** is the date that the Explain record has been added to the Explain database.
	• **dateChanged** is the date at which the Explain record has been last modified.
	• **expiry** states the date after which the Explain record content will be not any longer valid.
	• **humanString-Language** refers to the language applicable for descriptive text in the Explain record. In CIP the mandatory language is English, so no use is made of this feature.
	• the **otherInfo** provides additional information.

ASN.1 Definition	Meaning

`ContactInfo ::= SEQUENCE {`
`  name  [0] IMPLICIT InternationalString OPTIONAL,`
`  description  [1] IMPLICIT HumanString OPTIONAL,`
`  address  [2] IMPLICIT HumanString OPTIONAL,`
`  email  [3] IMPLICIT InternationalString OPTIONAL,`
`  phone  [4] IMPLICIT InternationalString OPTIONAL}`
	ContactInfo provides :**name**, **description**, **address**, **email** and **phone** of an organisation.
	In CIP **ContactInfo** is only used within **TargetInfo** to provide details about an organisation running a Retrieval Manager. Additional contact information can provided in each collection descriptors.

`HumanString ::= SEQUENCE OF SEQUENCE {`
`  language  [0] IMPLICIT LanguageCode OPTIONAL,`
`  text  [1] IMPLICIT InternationalString}`
	HumanString is used to provide message strings in different languages. For CIP at minimum English needs to be supported. **HumanString** contains:
	• **language** which specifies the language used for the text attached.
	• **text** is written in the language as specified in **language**.

`LanguageCode ::=InternationalString` `  -- from ANSI/NISO Z39.53-1994`	**LanguageCode** identifies (according to ANSI/NISO Z39.53-1994) what language is used.

`NetworkAddress ::= CHOICE {`
`  internetAddress  [0] IMPLICIT SEQUENCE {`
`  hostAddress  [0] IMPLICIT`
`  InternationalString,`
`  port  [1] IMPLICIT INTEGER},`
`  osiPresentationAddress  [1] IMPLICIT SEQUENCE {`
`  pSel  [0] IMPLICIT InternationalString,`
`  sSel  [1] IMPLICIT InternationalString`
`  OPTIONAL,`
`  tSel  [2] IMPLICIT InternationalString`
`  OPTIONAL,`
`  nSap  [3] IMPLICIT InternationalString},`
`  other  [2] IMPLICIT SEQUENCE {`
`  type  [0] IMPLICIT InternationalString,`
`  address  [1] IMPLICIT InternationalString}}`
	NetworkAddress can be specified by one of the following options:
	• an **internetAddress**, where the host address and port number are specified.
	• an **osiPresentationAddress**.
	• **other** network address specification.

ASN.1 Definition	Meaning
AccessInfo ::= SEQUENCE { -- AccessInfo contains the fundamental information about what facilities -- are required to use this target or server. For example, if an origin -- can handle none of the record syntaxes a database can provide, -- it might choose not to access the database. queryTypesSupported [0] IMPLICIT SEQUENCE OF QueryTypeDetails ~~OPTIONAL~~, diagnosticsSets [1] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER ~~OPTIONAL~~, attributeSetIds [2] IMPLICIT SEQUENCE OF AttributeSetId ~~OPTIONAL~~, schemas [3] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER ~~OPTIONAL~~, recordSyntaxes [4] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER ~~OPTIONAL~~, resourceChallenges [5] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER ~~OPTIONAL~~, restrictedAccess [6] IMPLICIT AccessRestrictions OPTIONAL, costInfo [8] IMPLICIT Costs OPTIONAL, variantSets [9] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL, elementSetNames [10] IMPLICIT SEQUENCE OF ElementSetName ~~OPTIONAL~~, unitSystems [11] IMPLICIT SEQUENCE OF InternationalString}	**AccessInfo** is used within **TargetInfo** and **DatebaseInfo**. If **AccessInfo** is used within **TargetInfo** it implies that at least one database at the Target will support the objects listed below. To determine which objects are supported for a particular database (collection), the client needs to retrieve the **AccessInfo** for that specific database (collection). **AccessInfo** provides: • **queryTypesSupported** which lists the different query types supported . • the **diagnosticsSets** supported. • **attributeSetIds** which identifies the attribute sets supported. For collections the provision of the object identifiers for **attributeSetIds** is mandatory. • all **schemas** supported. For collections the provision of the object identifiers for the **schemas** is mandatory. • all **recordSyntaxes** supported • all **resourceChallenges** supported. • **restrictedAccess** defines the access challenges**.** • **costInfo** describes the costs associated with certain operations, e.g. display of records. In CIP no costs are associated with search or display of records. The cost associated with ordering of data products are defined in the CIP Order Extended Services. • **variantSets** provides information about the variant sets supported. • **elementSetNames** lists all the element sets supported. • **unitSystems** lists all the unit systems supported.
QueryTypeDetails ::= CHOICE { private [0] IMPLICIT PrivateCapabilities, rpn [1] IMPLICIT RpnCapabilities, iso8777 [2] IMPLICIT Iso8777Capabilities, z39-58 [100] IMPLICIT HumanString, erpn [101] IMPLICIT RpnCapabilities, rankedList [102] IMPLICIT HumanString}	**QueryTypeDetails** provides a choice between different query types: • **private** , **iso8777**, **z39-58**, **erpn** and **rankedList** are not supported by CIP. • query type **rpn** is supported in CIP.
RpnCapabilities ::= SEQUENCE { operators [0] IMPLICIT SEQUENCE OF INTEGER OPTIONAL, -- Omitted means all operators are supported. resultSetAsOperandSupported [1] IMPLICIT BOOLEAN, restrictionOperandSupported [2] IMPLICIT BOOLEAN, proximity [3] IMPLICIT ProximitySupport OPTIONAL}	**RpnCapabilities** states what RPN capabilities are supported by CIP: • **operators** defines what operators CIP supports. CIP supports the 'and', 'or' and 'and-not' operators. • result sets are supported as operands in the CIP, thus the **resultSetAsOperandSupported** is always true. • **restrictionOperandSupported** is always false for the CIP. • **proximity** states whether proximity is supported or not. .

ASN.1 Definition	Meaning

`PerElementDetails ::= SEQUENCE {`
`name [0] IMPLICIT InternationalString OPTIONAL,`
`-- If the name is omitted, the record syntax's name for this element`
`-- is appropriate.`
`recordTag [1] IMPLICIT RecordTag OPTIONAL,`
`-- The record tag may be omitted if tags are inappropriate for the`
`-- syntax, or if the origin can be expected to know it for some`
`-- other reason.`
`schemaTags [2] IMPLICIT SEQUENCE OF Path OPTIONAL,`
`-- The information from the listed schema elements is in some way`
`-- to produce the data sent in the listed record tag. The 'contents'`
`-- element below may describe the logic used.`
`maxSize [3] IMPLICIT INTEGER OPTIONAL,`
`minSize [4] IMPLICIT INTEGER OPTIONAL,`
`avgSize [5] IMPLICIT INTEGER OPTIONAL,`
`fixedSize [6] IMPLICIT INTEGER OPTIONAL,`
`repeatable [8] IMPLICIT BOOLEAN,`
`required [9] IMPLICIT BOOLEAN,`
`-- 'required' really means that target will always supply the element`
`description [12] IMPLICIT HumanString OPTIONAL,`
`contents [13] IMPLICIT HumanString OPTIONAL,`
`billingInfo [14] IMPLICIT HumanString OPTIONAL,`
`restrictions [15] IMPLICIT HumanString OPTIONAL,`
`alternateNames [16] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,`
`genericNames [17] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,`
`searchAccess [18] IMPLICIT AttributeCombinations OPTIONAL}`
	PerElementDetails consists of:
	• **name** of the element.
	• **recordTag** of the element.
	• **schemaTags** which lists the schema elements that comprise the element within the record syntax.
	• **maxSize** which describes the maximum size of the element in bytes
	• **minSize** which describes the minimum size of the element in bytes
	• **avgSize** is the average size of the element.
	• **fixedSize** describes the size of the element in bytes, if the element has fixed length.
	• **repeatable** which states whether the element is repeatable.
	• **required** which states whether the element is required.
	• a human readable **description** of the element.
	• **contents** provides a description of the element
	• **billingInfo** defines specific billing associated with the element
	• **restrictions** describes the restrictions associated with the element
	• **alternateNames** which describes aliases under which the element is also known**.**
	• generic names for the element are defined in **genericNames**.
	• **searchAccess** which describes attribute combinations corresponding to this element.

`RecordTag ::= SEQUENCE {`
`qualifier [0] StringOrNumeric OPTIONAL,`
`-- E.g. tag set for GRS-1`
`tagValue [1] StringOrNumeric}`
	RecordTag provides for an element the following:
	• a **qualifier**
	• value for the tag in **tagValue**.

`AttributeCombinations ::= SEQUENCE {`
`defaultAttributeSet [0] IMPLICIT AttributeSetId,`
`-- Default for the combinations. Also probably a good choice for the`
`-- default in searches, but that isn't required.`
`legalCombinations [1] IMPLICIT SEQUENCE OF AttributeCombination }`
	AttributeCombinations states what combinations of attribute types are possible**.**
	AttributeCombinations consists of:
	• **defaultAttributeSet** which identifies the default attribute set to be used.
	• **legalCombinations** provides an array of all possible attribute type combinations allowed.

`AttributeCombination ::= SEQUENCE OF AttributeOccurrence` `-- An AttributeCombination is a pattern for legal combination of attributes`	**AttributeCombination** defines a list of all possible combinations of attributes types.

ASN.1 Definition	Meaning

`AttributeOccurrence ::= SEQUENCE {`
`  -- An AttributeOccurrence lists the legal values for a`
`  -- specific attribute type in a combination.`
`  attributeSet  [0] IMPLICIT AttributeSetId OPTIONAL,`
`  attributeType  [1] IMPLICIT INTEGER,`
`  mustBeSupplied  [2] IMPLICIT NULL OPTIONAL,`
`  attributeValues  CHOICE {`
`  any-or-none  [3] IMPLICIT NULL,`
`  -- All supported values are OK`
`  specific  [4] IMPLICIT SEQUENCE OF StringOrNumeric`
`  -- Only these values allowed`
`  }}`
	AttributeOccurrence list the legal values for a specific attribute type in a combination. It consists of:
	• an identifier of the **attributeSet**
	• an integer number , which identifies the **attributeType**
	• **mustBeSupplied** states whether the attribute type needs to be supplied.
	• **attributeValues** specifies the attribute values allowed in a combination**.**

`ElementInfo ::= SEQUENCE {`
`  elementName  [1] IMPLICIT InternationalString,`
`  elementTagPath  [2] IMPLICIT Path,`
`  dataType  [3] ElementDataType OPTIONAL,`
`  -- If omitted, not specified.`
`  required  [4] IMPLICIT BOOLEAN,`
`  repeatable  [5] IMPLICIT BOOLEAN,`
`  description  [6] IMPLICIT HumanString OPTIONAL}`
	ElementInfo provides for an element the following information:
	• **elementName** specifies the element's name
	• **elementTagPath** which provides the path for that element.
	• **dataType** which gives the type of the element and states whether the element is structured or not.
	• **required** indicates whether the element is required or not.
	• **repeatable** indicates whether the element is repeatable or not.
	• a human readable textual **description** of the element

`Path ::= SEQUENCE OF SEQUENCE{`
`  tagType  [1] IMPLICIT INTEGER,`
`  tagValue  [2] StringOrNumeric}`
	Path includes:
	• an integer for the **tagType.** In CIP the **tagType** is always 4.
	• a **tagValue** provides the tag value of the specific element**.**

`ElementDataType ::= CHOICE{`
`  primitive  [0] IMPLICIT PrimitiveDataType,`
`  structured  [1] IMPLICIT SEQUENCE OF ElementInfo}`
`  PrimitiveDataType ::= INTEGER{`
`  octetString  (0),`
`  numeric  (1),`
`  date  (2),`
`  external  (3),`
`  string  (4),`
`  trueOrFalse  (5),`
`  oid  (6),`
`  intUnit  (7),`
`  empty  (8),`
`  noneOfTheAbove  (100) -- see 'description'`
`  }`
	ElementDataType is one of the following:
	• **primitive** data type, where the integer value indicates the data type of an element (e.g. "date", "string").
	• **structured** data type, where a description of all the elements the element consists of is provided.

3.5.9.3 Searching the Explain Database

This section summarises issues of searching the *Explain database*. This is carried out by the *target* in response to a *request* from an *origin*.

Explain databases are accessed using standard Z39.50 search and retrieval services. Some search *terms*, corresponding to information categories, are predefined allowing a level of semantic interoperability. The '**exp-1**' *attribute set* includes *Use attributes* relevant for *Explain database* searching. Each **exp-1** *attribute* corresponds to a specific *element* in an *Explain record* and some *Explain attributes* correspond to *elements* that appear in more than one type of record (e.g. ***ExplainCategory*** and ***DatabaseName***) while others correspond to *elements* that appear in only one type of record.

The canonical search form is:

ExplainCategory = '*Category*'

AND

Identifier = '*Value*'

An example using ***TargetInfo*** is:

ExplainCategory = '*TargetInfo*'

AND

name = 'RetrievalManager_A'

The above example is of course a fragment of an *Explain* query that the Target would target at its *Explain database*. The fragment will be part of a *Type-1* query derived from a query initiated by a CIP *origin*.

If a *target* receives a *Search request* targeted at an *Explain database* which is not its own, the *target* determines the remote *target* that must handle the query (from the name of the *Explain database*, which contains the identifier of the *target* in which it is defined) and forwards the query to the remote *target*.

Combinations of ***exp-1*** *Use attributes* to perform a common set of searches are listed in Sections 3.2.10.1.1 and 3.2.10.1.4 of the Z39.50 specification[Z3950].

Names and descriptions of valid *target attributes* are provided in the ***AttributeSetInfo*** *Explain record*. Details of legal *attributes* for a target collection or product descriptor are listed in the ***AttributeDetails*** *Explain record* which also contains information on how those *attributes* can be combined in a single operand **-** ***attributeCombinations***.

Information can also be searched from the *Explain database* according to language, although there is no CIP requirement on language dependence, the ***HumanStringLanguage*** *element* can be used to locate descriptions in native languages. Information can also be searched according to control dates, in particular new records can be located using ***dateAdded*** and updated records using ***dateChanged***.

3.5.9.4 Retrieval from the Explain Database

To retrieve data from the *Explain database*, the *target* will specify the *Explain syntax* as the preferred *record syntax*. *Explain* information can be retrieved in summary known as *Brief* ('*B*') which will return those *elements* labelled *brief* or *Full* ('*F*') which will retrieve all *elements* (including *brief elements*). The *brief elements* together with a description (human readable text) can also be retrieved.

3.5.10 Termination Facility

The standard Z39.59 *Close service* procedures, described in Table 3-53 shall be followed.

Table 3-53: Close

ASN.1 Definition	Meaning

`Close ::= SEQUENCE{`
`  referenceId  ReferenceId ~~OPTIONAL~~,  -- See 3.2.11.1.5.`
`  closeReason  CloseReason,`
`  diagnosticInformation [3]  IMPLICIT InternationalString OPTIONAL,`
`  resourceReportFormat [4]  IMPLICIT ResourceReportId OPTIONAL,`
`  -- For use by origin only, and only on`
`  -- Close request; origin requests target`
`  -- to include report in response.`
`  resourceReport [5]  ResourceReport OPTIONAL,`
`  -- For use by target only, unilaterally`
`  -- on Close request; on Close response`
`  -- may be unilateral or in response to`
`  -- origin request.`
`  otherInfo  OtherInformation OPTIONAL}`
	A **Close** request/response allows an origin/target to abruptly terminate all active operations and to initiate the termination of the Z-Association. It contains the following information:
	• **referenceId**, which is the reference identifier of the operation.
	• **closeReason**, which describes the reason for the termination of the Z-Association.
	• **diagnosticInformation**, which is a text message providing additional information about the termination.
	• **resourceReportFormat**, which may be used by the origin to request that the target includes a resource report in the response. See section 3.5.6.3 for more detail.
	• **resourceReport**, which contains a resource report and may be included by the target when requested by the origin. See section 3.5.6.3 for more detail.
	• **otherInfo** about the Close request/response.

ASN.1 Definition	Meaning

`CloseReason ::= [211] IMPLICIT INTEGER{`
`finished (0),`
`shutdown (1),`
`systemProblem (2),`
`costLimit (3),`
`resources (4),`
`securityViolation (5),`
`protocolError (6),`
`lackOfActivity (7),`
`peerAbort (8),`
`unspecified (9)}`
	CloseReason describes the reason for the termination of a Z-Association. The reason for the termination may be one of the following:
	• **finished** indicates that the requester of the termination has finished its activity in the Z-Association.
	• **shutdown** indicates that the requester of the termination is shutting down.
	• **systemProblem** indicates that the requester of the termination has a system problem.
	• **costLimit** indicates that the requester of the termination has reached its cost limit.
	• **resources** indicates that the requester of the termination has exhausted its resources.
	• **securityViolation** indicates that the Z-Association is terminated because a security violation.
	• **protocolError** indicates that the Z-Association is terminated because of a protocol error.
	• **lackOfActivity** indicates that the Z-Association is terminated for lack of activity.
	• **peerAbort** inidcates that the Z-Association is terminated because a peer has aborted.
	• **unspecified** indicates that the reason for the termination of the Z-Association is unspecified.

3.6 Reference Identifier

Each *operation request* from a client (or Retrieval Manager) must have a unique *reference identifier*. This can be used in the tracking of an *operation* and is particularly important when a search is invoked which is passed onto remote Retrieval Managers (see Section 3.9.2) This means that the *request identifier* must indicate the starting *origin* of any *request* and also be unique within and beyond a single a *Z-association*, so must also indicate the date and time of the initiation of the *operation*. Finally a numeric identifier is included to identify a particular *operation*, in the situation where multiple *operations* are triggered at the same time.

The format for the reference identifier is described in Appendix E.

3.7 Concurrent Operations

The CIP supports the concept of *concurrent operations*, which means that a client can initiate multiple parallel *operations*, e.g. searches. In other words, a client does not need to wait for the termination of an operation (i.e. the reception of the response corresponding to a request) before initiating another operation (i.e. performing another request). In order to distinguish between the different operations, the CIP mandates the provision of a ***ReferenceId*** for every initiating *request* and that the *origin* (client or Retrieval Manager) assigns to each of the *concurrent operations* a different ***ReferenceId***.

Concurrent client *operations* shall be supported by the Retrieval Manager so that multiple *services* can be initiated simultaneously by the client. Concurrent *operations* are negotiated at *initialisation*. In the CIP, the choice of whether to use concurrent *operations* or not is made uniquely by the client. If the client *requests* that they be supported then the server must be able to *respond* affirmatively. Of course the client may be supported by a simpler user interface tool and hence only be able to support non-concurrent *operations*.

3.8 Diagnostic Messages

The Retrieval Manager shall use the *diagnostic format diag-1*, {Z39.50-Diagnostic-diag-1}. This *diagnostic format* permits the inclusion of defined *diagnostic conditions* such as the ones defined by bib-1 in {Z39.50-Diagnostic-bib-1} and the CIP *diagnostic conditions* defined in {Z39.50-CIP-Diagnostic}. The CIP *diagnostic conditions* are defined in Appendix F.

3.9 CIP Domain Specifics

This section specifies details of the protocol that are not generic Z39.50 Version 3 issues, but specific to the CIP domain. This primarily concerns CIP collections and their interaction with Z39.50 *facilities*. The subsections indicate how collections are intended to be managed and searched within the Z39.50/CIP domain and how the Retrieval Manager will interact with the Z39.50 *facilities* and collection tree.

The subsections include information on attribute/collection node retrieval (i.e. via the *Explain facility*), collection searching, product descriptor searching, item descriptor identification and collection member identification.

3.9.1 Attribute and Collection Node Retrieval

To facilitate effective searching and use of search *attributes* in order to query a particular item descriptor (i.e. collection or product descriptor), the client needs to understand the set of *Use attributes* which may be used to query that item descriptor. This set must, by definition, include the set of all the mandatory CIP *Use attributes* for the type of item descriptor (e.g. all the mandatory

collection *Use attributes* for a collection descriptor). Additionally, it may include a subset of the optional CIP *Use attributes* for the type item descriptor. Finally, for product descriptors, it may also include a set of local *Use attributes*, when local attributes are defined.
The search *attributes* which may be used to search a particular item descriptor are stored in the *Explain database* on a per collection basis and are described by semantic attributes. These search *attributes* can be searched using the *Search facility* by targeting a *Type-1* query at the *Explain database* in which the collection of interest is defined (i.e. the *Explain database* which is located at the same *target* than the collection- this can therefore be either the local or a remote *Explain database*). The search attributes can then be retrieved using the *Retrieval facility*[36]. If a *target* receives a *Search request* to query an *Explain database* which is not its own local *Explain database*, the *target* forwards the *Search request* to the appropriate *target* in order to query the remote *Explain database*. The same principle is applied for the retrieval of the data.

A generic collection tree hierarchy cannot be prescribed in this specification as the collection hierarchies and nodes will vary from site to site. The *Explain database* treats each collection node as a *target database* served by the Z39.50 server and therefore will maintain information about each collection node as it would any Z39.50 *target database*. This information would include, in the ***DatabaseInfo*** category of the *explain database*, an indication of which collection node was the 'root' collection and also key access nodes for the collection hierarchy (see Section 3.5.9.2.3 for details). The 'root' collection node, identified with the *target database* name '***IR-Collection-1'***, has no local parent and effectively encompasses all collections owned by a Retrieval Manager.
This information can be retrieved at an appropriately convenient stage following *initialisation*[37].

3.9.2 Collection Searches

A collection search is a Z39.50 *Type-1* search that is used to identify and retrieve collection definitions. Each collection is defined by a set of attributes that are used to describe the collection members as a whole. The values of these attributes are analysed in the search against the search criteria. By targeting a search at a collection, the Retrieval Manager will automatically search all collections below the target collection, i.e. those that are included hierarchically in the collection tree rooted by the target collection.

To perform a collection search, the client targets the search at the collections *database* '***IR-Collection-1'*** (i.e. the root collection) and identifies the specific collection below[38] which the collection search must be performed within the search criteria specified in the query[39].

---

[36] The method and timing of the retrieval of this *attribute* definition information is outside the scope of this specification, but options include:

- direct retrieval by the client from the *Explain database*, of all *attributes*, immediately after initialisation;
- retrieval by the client via the Retrieval Manager at an appropriate stage of the processing;
- traversal of the collection nodes by the Retrieval Manager and retrieval of appropriate *attributes*;
- ad-hoc retrieval of *attributes* as needed by the client.

This raises issues such as caching, automatic updating, etc., but these are design and implementation concerns.

[37] A generic Z39.50 version 3 client which does not support the *Explain facility* may compile an appropriate list of all the CIP mandatory *Use attributes* and the appropriate CIP optional *Use attributes* from Appendix A in order to be able to access CIP servers.

[38] Having a separate collection *database* as a named *database* (***IR-Collection-1***) means that it itself has an entry within the *Explain database*, and that it could therefore specify its own *relation attributes* such as 'parent', 'sibling'. These 'navigational' relations may be considered for addition in further releases.

For example, if a client wants to perform a collection search targeted at the collection 'Collection 22', the following definitions would be included in the *Search request*:

target database: ***IR-Collection-1***

RPN Query: TemporalCoverage = 'Term 1' AND SpatialCoverage = 'Term 2'AND (CollectionDescriptor/ItemDescriptorId Included in 'Collection 22')

There are a number of important assumptions on collection hierarchies and on the Retrieval Manager that are required for effective collection searching:

- collections within a collection tree are defined by the same set of collection descriptor attributes;
- not all collection descriptor attributes for a particular collection node need to have values;
- collections underneath a particular collection node (i.e. subordinate collections) logically belong in that part of the hierarchy, whether by virtue of their common attributes or by virtue of their attribute values, (i.e. logically consistent collection hierarchies are built);
- a number of collection descriptor attributes are defined so that if they are included in a search *term* and a collection node fails to match the search criteria for that search *term*, then any subordinate collections would also fail to match the search criteria (labelled consistency attributes)

When the above assumptions are taken into account, then more efficient collection hierarchy searching is feasible.

If a successful match results from searching the attributes of a collection, then the Retrieval Manager will forward the search to the entire collection structure below. Note a collection search shall not be executed on collection members which are terminal collection members (product descriptors). If the collection members are collections owned by another Retrieval Manager, then the search shall be passed onto that Retrieval Manager by the originating Retrieval Manager acting as an *origin* to the new *target* Retrieval Manager. This shall continue until the search is complete.

To identify if a search has already been executed on a collection, each search that is forwarded on to another Retrieval Manager, must include the ***ReferenceId*** established for the original search *operation* (see Section 3.6 for information on ***ReferenceId***) in the ***additionalSearchInfo*** field of the search object (see Section 3.5.2.1 for information on search object elements). This ***ReferenceId*** is unique for any *operation* within and beyond a single *Z-association* so that a remote Retrieval Manager can log which particular searches have 'visited' a collection and if a 'visit' has already occurred the remote Retrieval Manager can respond immediately with an appropriate *response* (effectively an 'AlreadySearched' flag). The search *response* will also include an appropriate Z39.50 *diagnostic message*, rather than just a response of 'zero matches' as zero matches could be interpreted as a successful search with no matches.

A remote Retrieval Manager shall coordinate searches of all collection that it owns; this means that the results of a search shall be assembled by the owner Retrieval Manager of all collections within that single remote Retrieval Manager. The *result set* from the remote Retrieval Manager is then returned to the originating Retrieval Manager to further compile into a single *result set* (i.e. there is not a *result set* per collection within a single Retrieval Manager domain).

[39] If a collection search is not targeted at a particular collection in the collection hierarchy (by the specification of the collection below which the search should be performed in the search criteria), the root collection IR-Collection-1 is searched and thus all collections are effectively searched.

3.9.3 Product Descriptor Searches

A product descriptor search is a Z39.50 *Type-1* search that is used to identify and retrieve product descriptors.

For a product descriptor search the user selects first a collection from which he wishes to search all product descriptors that are contained in the terminal collections below the selected collection at which the search is targeted. The Retrieval Manager searches then down the collection tree for the terminal collections (collection search) and identifies those product descriptors that are within the terminal collections. The search is then directed to the relevant product descriptor database manager (i.e. local catalogue inventory system). If a collection in the collection tree includes a remote collection, then the complete search is passed to the remote Retrieval Manager identified, targeting the remote collection and the same procedure will be followed at the remote Retrieval Manager. The remote Retrieval Manager will return a *result set* to the original Retrieval Manager, which will then compile the results from all sources and make the *result set* available to the client.

After a product descriptor search, a client might want to obtain additional information (e.g. contact details) which is stored in the collection descriptor of the terminal collection to which the product descriptor belongs.

To identify the collection for which the collection descriptor needs to be searched, the client uses the collection path that is returned in the ***DatabaseName*** field of the product descriptor *search response* object (see Section 3.5.2.6.2). The last collection in the collection path is the terminal collection.

In the case that the collection is a theme collections, additional information about a product descriptor can be found in the archive collection descriptor from where the product descriptor originated. This archive collection can be referenced by the RelatedCollectionDescriptors (sub-element of the RelatedItemDescriptor) in the product descriptor.

3.9.4 Collection Path Identification

The results from either a collection or a product descriptor search shall include in the ***DatabaseName*** field of the presented object, automatically the full collection tree path from the original target collection of the search to the item descriptor located (whether it be a collection descriptor or product descriptor). A client might wish to optionally return the full collection path to the user as an additional source of information, e.g. the user might use the information to minimise future search times. Another use of returning the full collection path might be that a client uses this information to look for duplicate result records. The hierarchical information provided by the full collection tree path enables the client to customise the handling of duplicates according to its specific needs (i.e. the choice of one duplicate over another may be relevant for a particular client, for example, for reasons of ordering cost). This provides an extra level of service to a local catalogue.

That is, the identification of the *target* Retrieval Manager, followed by the identifiers of the collections down to the point in the collection tree where either a terminal collection or a remote collection is encountered. If a remote collection is encountered, then there follows a double solidus ('//') plus the identifier of the remote Retrieval Manager, this itself is followed by the collection tree path in the remote Retrieval Manager, again until a terminal or remote collection is encountered. (For the definition of a single collection identifier, (i.e. not in the format returned in a *search response*, which includes the path), refer to Section 3.5.2.3: **Database Names**).

For example, consider a collection tree hierarchy as shown in Figure 3-22:

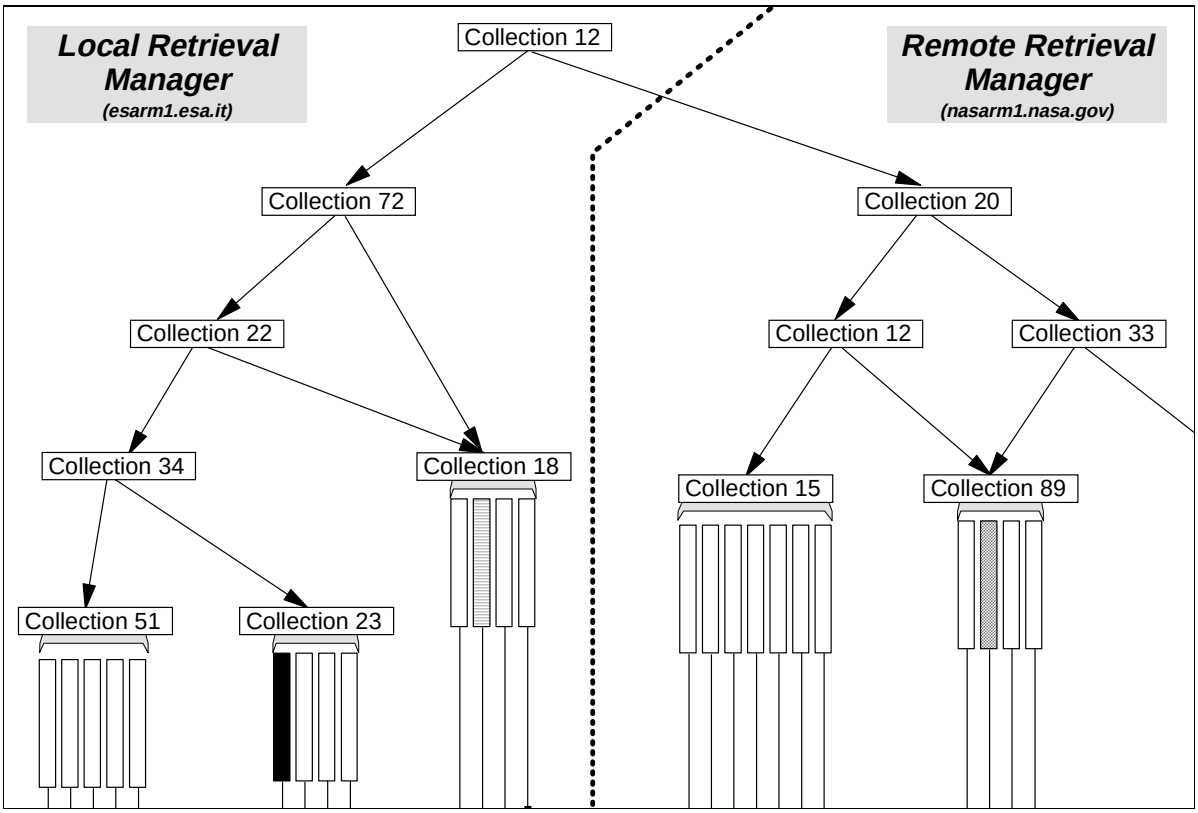

Figure 3-22: Example Collection Path Identification

If a search was targeted at collection 12, which included in its results the product descriptor in solid black, the **`DatabaseName`** field would be **`/esarm1.esa.it/12/72/22/34/23`**. The same search targeted at collection 22 would result in **`/esarm1.esa.it/22/34/23`**.

If the search was targeted at collection 12 and resulted in identifying the product descriptor with horizontal shading, there would be two result records (as the product descriptor is found via two collection tree routes), these would have **`DatabaseName`** fields of **`/esarm1.esa.it/12/72/18`** and **`/esarm1.esa.it/12/72/22/18.`**

Finally, if the search was targeted at collection 12 and resulted in identifying the product descriptor with chequered shading, this would also result in two result records, with the **`DatabaseName`** fields of **`/esarm1.esa.it/12//nasarm1.nasa.gov/20/12/89`** and **`/esarm1.esa.it/12//nasarm1.nasa.gov/20/33/89.`**

An exact definition of the collection path to be used for the ***DatabaseName*** is provided in Appendix E.7.

3.9.5 Collection Member Identification

To enable a client to retrieve a complete definition of the collection tree below a target collection, there is an *element set name* (i.e. standard result format, see Section 3.5.3.3) defined to convey this information in a standardised manner.

The client shall target a search at a collection, with a *Use attribute* of **`CollectionId`** and a *term* (i.e. search value) of the same value as the target collection identifier. This will mean that only the target collection will be matched in the search and no other collections. The *result set name*, requested in the

search, only includes the collection identifier plus the identifiers of the members of the collection (see Section 3.9.4), therefore the *origin* is returned only the members of the target collection[40].

This process can be repeated for each of the members of the collection returned and hence in this way the complete collection tree can be established.

Another way of establishing the complete collection tree would be to perform the same search but with the **ANY** Use attributes, this would result in all collections being matched and therefore the complete collection tree returned in one single operation.

A formal specification of the Collection Member identification, according to Z39.50, is given in Annex D.

3.9.6 Handling of Local Attributes

The CIP attributes defined in this document (see the definition of the searchable CIP attributes in Appendix A, the definition of the CIP *elements* in Appendix B and the CIP *schema* in Appendix C) provide a high-level of interoperability between Retrieval Managers as common attributes are shared within the CIP space. However, an agency may wish to extend the core metadata defined for a product descriptor (as provided by the CIP attributes) with additional attributes in order to fulfil its own needs. This is achieved by the definition of local attributes.

The definition of local attributes is out of the scope of the CIP profile. However, the CIP profile provides sufficient flexibility so that local attributes can be easily and freely added. This section provide the definitions and guidelines to allow an agency to define its own local attributes in compliance with the CIP profile.

3.9.6.1 Introduction

In addition to the CIP attributes defined in this specification, each agency may define his own local attributes that are available at his Retrieval Manager. An agency is free to define local attributes according to its own needs. The only restrictions imposed by the CIP profiles are the following:

- Local attributes must be defined only for product descriptors, i.e. no local attribute must be defined for collection descriptors.
- Local attributes must be defined on a per-collection basis. The definition of the local attributes must be provided at the level of the archive collection which owns the product descriptors for which the local attributes are defined[41].
- Local attributes may be defined only when no appropriate CIP attribute is available. In other words, if a provider's product information can be mapped into a standard CIP attribute, this information must not be held by a local attribute.
- Each local *database schema*, including the local attributes, must be compliant with the CIP *database schema* and must therefore contain all the mandatory CIP *schema elements*. It this way, each local *database schema* must be a superset of the CIP schema.

---

[40] The response to this search can be returned by any Z39.50 standard method, i.e. the *Present service* or the *piggyback* feature, this is at the discretion of the application. As the members of a collection are expected to be relatively small in number, this would be a convenient place to use the *piggyback* feature of the *search response*.

[41] In other words, the local attributes definitions must be provided in the collection descriptor of the collection which owns the product descriptors for which the local attributes are defined, whereas the actual local attributes are included in the product descriptors themselves. This is valid independently of the method used for the definition of the local attributes, i.e. either directly in the archive collection descriptor or in the *Explain* definitions related to the archive collection descriptor.

The scope of local attributes is restricted to the domain of the Retrieval Manager which defines them rather than the whole CIP domain (as for the CIP attributes) and therefore no interoperability can be assumed for local attributes. For instance, if a search query contains a local *use attribute*, this local *use attribute* will be applicable to (i.e. will be recognised by) only the collections owned by the Retrieval Manager which defined the local *use attribute*. Note that queries with local attributes must only be directed to archive collections.

The CIP supports two different methods to define local attributes:

- **Local attributes via Explain:** the local attributes are defined and used in exactly the same way as the CIP attributes. They are defined in (and their semantics can be queried from) the *Explain database*. This method is presented in more details in Section 3.9.6.3.
- **Local attributes in Collection Descriptor:** the local attributes are defined within (and their semantics can be queried from) the collection descriptor which owns the product descriptors for which they are defined . With this method, the local attributes are therefore defined and queried differently than the standard CIP attributes. This method is presented in more details in Section 3.9.6.4.

Both approaches are currently valid and the choice of the method used is up to the agency which defines the local attributes. Moreover, as the two methods are not mutually exclusive, a Retrieval Manager could support both methods. However the Explain database is the preferred solution and is considered, in case of conflict, as the reference. Note that in future, after more information is available from implementations, this situation could change and only one solution might be supported.

Whilst the local attributes' definition and semantics can be defined via two different methods, the retrieval of the local attributes in the product descriptor (i.e. the local *schema elements*) is always performed in the same manner, by requesting the 'Local Attributes' or the 'Full' *element set* in a *Present request* for the product descriptor.

3.9.6.2 Local Objects Definition

Independent of the chosen method, an agency who wishes to define its own set of local attributes will need to define, at least[42], the following Z39.50 locally registered objects:

- A local *attribute set*.
- A local *tag set*.

Additionally, if the local attributes are defined via Explain the following Z39.50 locally registered object need also to be defined:

- A local *database schema* (which will integrate the CIP *schema elements* with all local *schema elements* available at the Retrieval Manager).

It is recommended that the local attributes are defined according to the models provided in Appendices A, B and C.

The Z39.50 locally registered objects defined by an agency must be registered in the same way as the CIP registered objects and use OIDs as specified in the Z39.50 specification Appendix 'OID.6 Locally Registered Objects', which have the form:

[42] An agency could define, and maintain, several *attribute sets*, *tag sets* and local *schemas*.

{ANSI-standard-Z39.50 n 1000 p m}

where: n = the class of the object ('3' for *attribute sets*, '13' for *database schemas* and '14' for *tag sets*);

p = the OID index of the agency as a Z39.50 registered implementor;

m = number of the object being defined.

3.9.6.3 Local Attributes via Explain

The section introduces how local attributes may be defined in the Explain database, how the local attributes' semantics may be retrieved from the Explain database and how the local attributes in a product descriptor may be retrieved.

3.9.6.3.1 Local Attributes Explain Definition (Explain)

When local attributes are defined via Explain, the local attributes are managed by the Retrieval Manager in the same way as the CIP attributes (see Section 3.5.9 for more details).

As each collection is a *database* as far as CIP is concerned, it shall therefore contain the description of the local *database schema*, *tag set* and *attribute set* in its entries in the *Explain database*. Each collection which needs to use local attributes shall use a local *database schema*[43] (instead of the CIP *database schema*). The local *elements* which are necessary to define the local *schema* need to be defined in a local *tag set*. A local *tag set* can be freely defined by each agency. The only restriction is that the local *tag set* does not use the *tag type* "4", which is reserved for the CIP *tag set*[44]. The local *schema* must be a compatible superset of the CIP *schema*. The following conditions must be met:

- All the mandatory CIP *schema elements* must be included in the local *schema*, i.e. no mandatory CIP *schema element* can be removed.
- New local *schema elements* may be added in the local *schema*. The local *schema elements* may be plugged-in anywhere in the original CIP *schema*, e.g. a leaf *schema element* may be added in an existing compound, a new compound may be inserted.
- An optional CIP *schema element* may be mandatory in the local *schema*. However, a mandatory CIP *schema element* cannot be made optional in the local *schema*.

Note that new *schema elements* will mainly be defined from local *elements* (defined in the local *tag set*). However, CIP *elements* may also be used to create local *schema elements* (e.g. the CIP *elements* 'PersonName' could be re-used in a local compound *schema element* if appropriate).

The *elements* defined in the local *tag set* which are searchable in the local *schema* need to be defined in the local *attribute set*. The local *attribute set* is therefore a sub-set of the local *tag set*.

3.9.6.3.2 Search of Local Attributes Semantics (Explain)

The meaning of each local attribute can be queried, and retrieved by a client or a remote Retrieval Manager by querying the *Explain database*. Similarly, the *attribute combinations* involving the local attributes are also contained in the *Explain database*. This enables a client (any client in the CIP domain) to retrieve the semantics of local *use attributes*, and the semantics of the queries that can be

[43] It is important to note that local attributes are used in a per collection basis and that therefore different collection on the same Retrieval Manager may use different local *database schemas* (i.e. different local attributes). However, several collection may share the same local *schema*. In fact, a single local *schema* may be defined for all the collections using local attributes at a Retrieval Manager.

[44] Note that there is no restriction on the assignment of *tag values* to the local *elements* as each *element* (CIP or local) is identified with a *tag type*/*tag value* pair.

performed with them. This, in turn, enables any client to perform search queries containing local *use attributes*.

3.9.6.3.3 Local Attributes Retrieval (Explain)

The retrieval of records containing local *schema elements* is performed in the same way as the retrieval of records containing CIP *schema elements*, which is fully described in Section 3.5.3. The only differences are:

- for the databases containing local *elements*, a local *database schema* can be used (i.e. a local *abstract record structure* is applied) instead of only the CIP *database schema*. Technically, this is achieved by using a ***dbSpecific*** *database schema* instead of the ***generic*** CIP *database schema* in the *composition specification* ***CompSpec*** (see the formal definition in Section 3.5.3.5.1 for more details). Note that, as a local *database schema* is, by definition, a superset of the CIP *database schema*, all the information that are mandated by the CIP specification will automatically be retrieved when a local *database schema* is used.
- the local *schema elements* can be requested during product descriptor retrieval by specifying that either the "Full" *element set* or, if only local *schema elements* are wanted, the "Local Attribute" *element set* is desired.

3.9.6.4 Local Attributes via Collection Descriptor

The section introduces how local attributes may be defined in the collection descriptor which owns the product descriptors for which local attributes are defined, how the local attributes' semantics may be retrieved from the collection descriptor and how the local attributes in a product descriptor may be retrieved.

3.9.6.4.1 Local Attributes via Collection Descriptor Management

When local attributes are defined via the collection descriptor, the local attributes are managed by the Retrieval Manager in a different manner than the CIP: instead of being stored in the Explain database, the local attributes are stored directly in the collection descriptor which owns the product descriptors for which the local attributes need to be defined.

When defining local attributes directly in the collection descriptor, the following restrictions apply compared with the definition of local attributes in the Explain database:

- All the local attributes defined in the collection descriptor are searchable, i.e. there is a one to one mapping between local use attributes definition in the collection descriptor and the local schema elements definition in the product descriptor.
- Local attributes are defined as a flat list of leaf local attributes included in the CIP schema in a reserved space (i.e. the LocalProductUserAttributes compound, see Appendix C for more details). As a consequence, the following limitations apply:
  - No compound local attributes may be defined.
  - Local attributes cannot be "plugged-in" existing CIP compound attributes.

A local *tag set* and a local attribute set needs to be defined for the Retrieval Manager[45]. However, the semantics of the elements and use attributes need not to be defined in the Explain database as they are explicitly described in the collection descriptor.

[45] At least one *tag set* and one *attribute set* need to be defined. However, an agency may define more than one *tag set* and *attribute set*.

3.9.6.4.2 Search of Local Attributes Semantics (Collection Descriptor)

The meaning of each local attribute can be queried, and retrieved by a client or a remote Retrieval Manager by querying the collection descriptor which owns the product descriptors for which local attributes are defined. This is achieved by performing a Search request on the collection descriptor and retrieving the local attributes definition with a Present request, using the "Local Attributes" element set. As a result, the list of all the local attributes, together with their semantics, is obtained. The local attributes can then be used to search the product descriptors.

3.9.6.4.3 Local Attributes Retrieval (Collection Descriptor)

The retrieval of records containing local *schema elements* is performed in the same way as the retrieval of records containing CIP *schema elements*, which is fully described in Section 3.5.3. The local *schema elements* can be requested during product descriptor retrieval by specifying that either the "Full" *element set* or, if only local *schema elements* are wanted, the "Local Attributes" *element set* is desired. Note that no local database schema needs to be specified as the local attributes are included in a reserved space in the CIP product descriptor schema (i.e. the LocalSchemaElements compound schema element), which content is dynamically determined during the retrieval.

3.9.7 Diagnostics Usage during distributed Search and Retrieval

In this section, details are provided on how *non-surrogate* and *surrogate diagnostics* are used during distributed searching (i.e. in *SearchResponse*) and retrieval (i.e. in *PresentResponse).*

This is described in the form of the following three scenarios which involve a CIP *Search/Present Request* targeted at two remote collections (collections A and B) which are managed by remote Retrieval Managers:

- **Case 1**: Both collections can process the request and do so successfully, finding 0 or more matches.
- **Case 2**: One Collection can process the *Search/Present* (i.e. collection A), whereas the other one cannot (i.e. collection B).
- **Case 3**: Neither collection can process the *Search/Present*.

The results provided for each case in the *SearchResponse* are presented in Table 3-54[46], and the results provided for each case in the *PresentResponse* are presented in Table 3-55. Note that, in both cases, the fields which are not impacted are not listed.

[46] The scenario presented assumes that no *piggybacking* was requested.

Table 3-54: Use of diagnostics in SearchResponse

Fields in SearchResponse	Case 1	Case 2	Case 3
resultCount	Number of records from collection A + Number of records from collection B	Number of records from collection A	0
numberOfRecordsReturned	0	1	1 or 2 as appropriate
nextResultSetPosition	1 (or 0 if no records matched either from collection A or B)	1 (or 0 if no records matched from collection A)	0
searchStatus	success	failure	failure
resultSetStatus	not set	subset (thus meaning that Present requests can be issued)	none
presentStatus	as required	not set	not set
records	No records (no piggybacking is requested and no diagnostics should be returned)	1 non-surrogate diagnostic for collection B	1 (All failed) or 2 (collection A failed, collection B failed) non-surrogate diagnostics

Table 3-55: Use of diagnostics in PresentResponse

Fields in PresentResponse	Case 1	Case 2	Case 3
numberOfRecordsReturned	Number of requested records from A + Number of requested records from B	Number of requested records from A + 1 or more (surrogate diagnostics for collection B)	1 or 2 as appropriate
nextResultSetPosition	Next record to be presented of 0 if result set is finished	Next record to be presented of 0 if result set is finished	0
presentStatus	success	failure	failure
records	Records from collection A + collection B	Records from collection A + 1 or more (surrogate diagnostics for collection B)	1 (All failed) or 2 (collection A failed, collection B failed) non-surrogate diagnostics

3.9.7 Security

This section describes how the Z39.50 *Access Control facility* is used to support the authentication and security requirements for the CIP. The requirements are described in the ICS URD[URD] and a discussion of the issues is presented in the Technical Note on Security [STN].

The CIP security requirements centre around the need to ensure an appropriate level of confidence in the identity of a user (or Retrieval Manager) such that access to services and functions can be controlled across the ICS. The key function identified as requiring authentication of users is ordering, however other operations may also require authenticated access. The access rights given to user groups is the responsibility of the data provider. The ethos of the ICS is that the system must be open to searches from users, even "guest" users in order to maximise the benefits of interoperability. However there is a need to protect commercial interests of data providers and to maintain the operational integrity of the ICS elements.

The CIP provides the capability to transfer authentication information between *target* and *origin* pairs of a Z39.50 association. This means that CIP clients can exchange authentication information with local Retrieval Managers, and Retrieval Managers can exchange authentication information with other Retrieval Managers.

The level of security that the transfer of this information provides is dependent upon the observation of good security practises by users, system managers of ICS elements and the implementors of Retrieval Managers and client applications.

3.9.7.1 Authentication

Authentication is the process of identifying a user or Retrieval Manager to an appropriate level of confidence. CIP B supports authentication through digital signatures. The structure used to exchange the signature information is the *encrypted* structure. The *encrypted* structure is exchanged in *AccessControlChallenge* and *AccessControlResponse* messages.

A digital signature is the result of applying a cryptographic algorithm, together with a key, on message. This forms a unique "signature" for the message and key. The methods of generating the signature and for the receiver to check the signature differ according to whether symmetric key or public key cryptosystems are being used. They may also differ between two symmetric key systems or two asymmetric key systems. For public key systems, it is also possible for the authenticity of a user to be checked through the use of a trusted third party or certification authority.

The CIP security approach is to allow a target to challenge an origin at any point during the session. In particular, the Retrieval Manager can request that any operation which requires authentication be signed by the user to support non-repudiation. The signed request can then be stored by the *target* as a "proof[47]" that a specific user requested the service.

There are a number of publicly available cryptographic algorithms which can be used to provide authentication and ensure the integrity of information transferred. These can be used to provide different degrees of security from moderate to public key cryptographic systems of near military quality. An important consideration for CIP is the trade-off between the degree of complexity passed onto the user and the complexity of the system through the implementation of security features.

The CIP provides the capability to support symmetric and asymmetric security systems. CIP compliant Retrieval Managers must support the symmetric key approach and optionally may support

[47] Note that the law on network access to services is still very much in its infancy, and the situation is particularly confused when the *target* and *origin* span different legal systems (as will often be the case with the ICS). "Proof" here depends on the local and remote legal interpretation of the security of the algorithm used, the key mechanism used, the system used to store the information etc.

the asymmetric key approach. It is the responsibility of the provider of a Retrieval Manager to ensure that a security approach is adopted which is appropriate for the data and services provided to its CIP users. A given Retrieval Manager may support more than one approach. For example, users may be supported using symmetric key authentication and other Retrieval Managers may be supported through public key authentication. The approach taken and confidence associated with each relationship (user - Retrieval Manager, Retrieval Manager - Retrieval Manager) is the responsibility of the organisation providing access to the Retrieval Manager.

The CIP client(s) which are used to access the Retrieval Managers also require the ability to run the authentication software (and to use the software in the manner prescribed by the CIP B spec and the security procedures of the local Retrieval Manager. It is suggested that CIP clients are implemented so that the authentication elements are easily configured and changed to allow the client to be configured for the relevant local Retrieval Manager security policy.

3.9.7.2 Shared (Symmetric) Keys

For the purposes of the ICS, a shared (or symmetric) key mechanism provides a minimum level of security for authentication purposes. The shared key will be used to provide a Message Authentication Code which acts as the digital signature for communications between a user (or Retrieval Manager acting on behalf of a user) and a Retrieval Manager. This approach allows the user to provide authentication information to the Retrieval Manager on demand, which the Retrieval Manager can compare with what it expects from the user. Both the Retrieval Manager and the user keep an identical copy of the key.

In symmetric key systems, a digital signature (MAC) is produced by supplying a key and data to a special function. The function (a one way hashing function) produces a result string which is the unique result for the data and key provided. The receiver of a signed message compares the signature provided with the result of applying the function and user key to the body of the message. If the two result strings are equal then the signature is confirmed. Since it is assumed that only the user has the other copy of the key, only the user can have signed the message. In addition, the MAC guarantees that the message has not been tampered with since the result of applying the signature depends on the complete message. A MAC therefore provides a check for data integrity. Figure 3-23 presents this process.

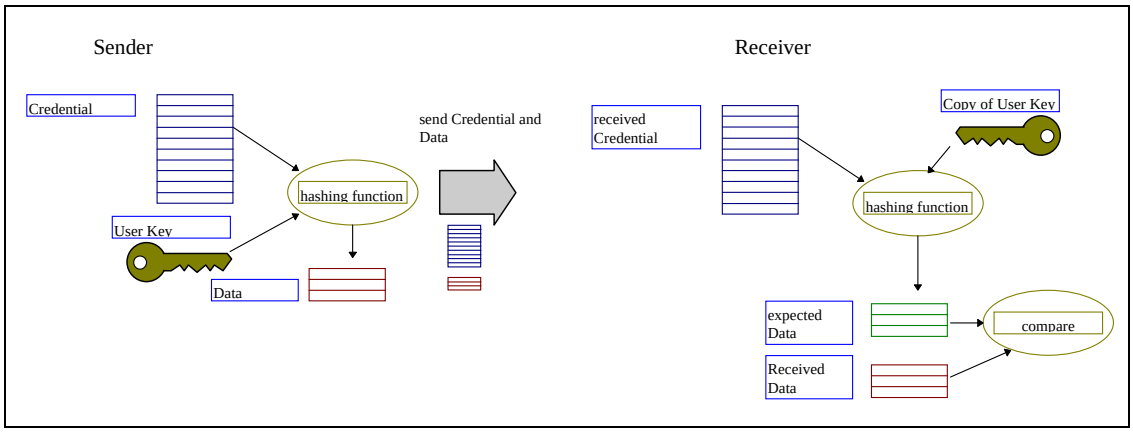

Figure 3-23: Symmetric Key Authentication Process

The hashing function for shared key approaches in the CIP B Specification is MD5. Sources for this algorithm are publicly available in the 'C' language and can be compiled for use on personal computers and workstations. The specification of the algorithm itself (RFC 1321) contains the source code.

When challenging an *origin*, the *target* supplies a set of parameters to be signed by the *target*. When authenticating a *target*, the *credential* element of the *encrypted* structure contains the information the

target wants the *origin* to sign. The uniqueness of this data ensures that the *AccessControlResponse* cannot be captured by a third party and subsequently replayed. Uniqueness is provided through the inclusion of a timestamp in the *credential* information.

For symmetric keys, the format of the *encrypted* structure (and hence the parameters used as an input to the hashing function) in an *AccessControlRequest* is:

crypt type = "MAC-MD5"

credential = {Timestamp, UserId, [Operation], RandPad, Timestamp}

data = NULL

Where

"UserId" is the user identifier against which the user is being authenticated. For pass-through authentication (see Section 3.9.7.8), the user identifier must be preceded by the identifier of the remote Retrieval Manager at which the user is authenticated[48].

"Operation" is the complete CIP operation being authenticated. If the CIP Order ES Request is being authenticated then this ASN.1 structure (and its content as received by the target) is repeated in the credential, so that the original order request details will be returned in the credential. This is to ensure that the signed operation can be used for non-repudiation purposes. Note that the CIP client must provide some form of notification to the user that a particular *operation* is being signed in order to prevent misuse of this *facility*. "Operation" is required only when non-repudiation is desired (i.e. for an order submission request). Otherwise, "Operation" does not need to be included in the credential.

"RandPad" is a randomly generated sequence of data produced by the *target* to pad out the credential

"Timestamp" is a Retrieval Manager (*target*) generated timestamp.

The format of the *encrypted* structure for the *AccessControlResponse* is:

crypt type = "MAC-MD5"

credential = {Timestamp, UserId, [Operation], RandPad, Timestamp}

data = H_{MD5}{SharedKey, Timestamp, UserId, [Operation], RandPad, Timestamp, SharedKey}

Where

"SharedKey" is the shared symmetric key, other parameters are as for the *AccessControlRequest*, except that "Operation" is provided only when non-repudiation is mandated for the operation (e.g. the submission of an order).

H_{MD5}{}is the result of applying the MD5 hashing algorithm to the content described in the braces.

3.9.7.3 Asymmetric (Public) Key

Asymmetric key cryptography uses two keys: one key to encrypt a message and another to decrypt the message. The two keys are mathematically related so that data encrypted with one key can be decrypted with the other, but there is no way to determine one key from knowledge of the other. The approach depends on the relationship where:

PrivateKey(PlainText) = CipherText and **PublicKey**(CipherText) = PlainText, and
PublicKey(PlainText) = CipherText and **PrivateKey**(CipherText) = PlainText

[48] This must be indicated so that the client knows that the user identifier authenticated is the user identifier at the remote Retrieval Manager rather than the Retrieval Manager with which the user has directly established a Z-Association.

In the CIP context, a user will have a private key and a public key. The public key will also be held by the local Retrieval Manager. Each Retrieval Manager will also have a key pair. The principle of public key cryptosystems is that the user can sign message using their private key and the resulting encrypted message can only be decrypted using the user's public key. Since only the user has access to their private key, only the user can have signed the message.

Public key cryptographic systems are considered to be much stronger than symmetric key systems since the private key is never distributed. The security of a public key system depends on the key management process. It is essential that the parties can get a copy of each other's public key from a completely trustworthy source. It is also essential that the private key is not compromised.

Figure 3-24 shows a simple form of authentication using a public key cryptographic system.

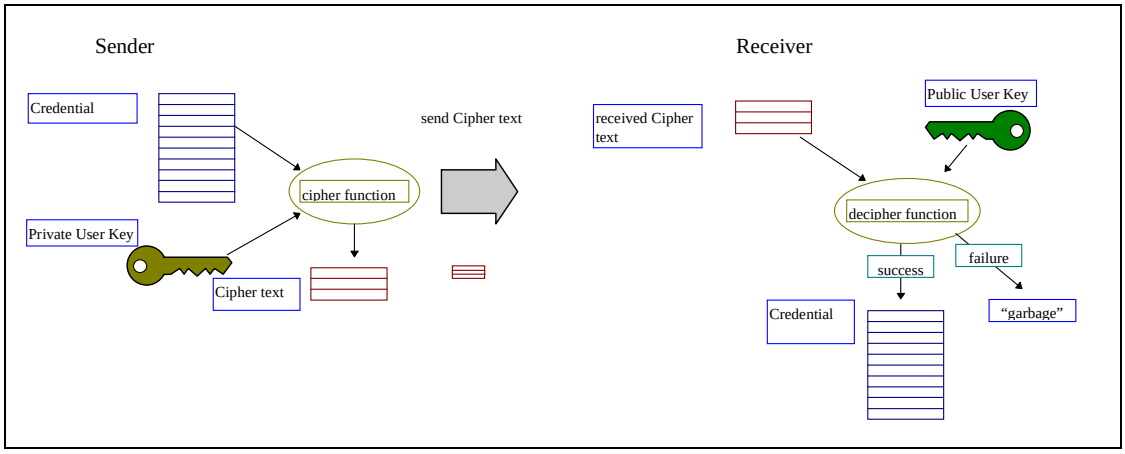

Figure 3-24: Simple Asymmetric Key Authentication

In the process represented in the diagram, the whole message is encrypted using the private key and decrypted using the public key. However the public key cryptographic algorithms are slow to run and the transfer of encrypted data over international networks using strong cryptographic algorithms can have legal complexities. An alternative approach is adopted in CIP B as follows:

1) A hash of the original message is produced using the MAC-MD5 algorithm and a randomly generated key. The result of applying the MD5 algorithm is a "digest" of the original message which is also typically much smaller than the original message.

2) The digest and randomly generated key are then encrypted using the private key of the user - this uniquely identifies the user and forms a digital signature.

3) The Original (clear text) message, encrypted digest and encrypted key are sent to the Retrieval Manager.

On reception the Retrieval Manager performs the following actions:

1) Decrypts the key and digest using the public key of the user.

2) Calculates the message digest of the original message using the decrypted key.

3) Compares the resulting digest with that produced from decrypting the passed digest.

The purpose of calculating the hash of the original message is two fold: first it provides a compressed representation of the message; and secondly it avoids any problems associated with the use of strong encryption algorithms to transfer enciphered data across an international network.

Figure 3-25 depicts the process of authentication in CIP B using a public key cryptosystem.

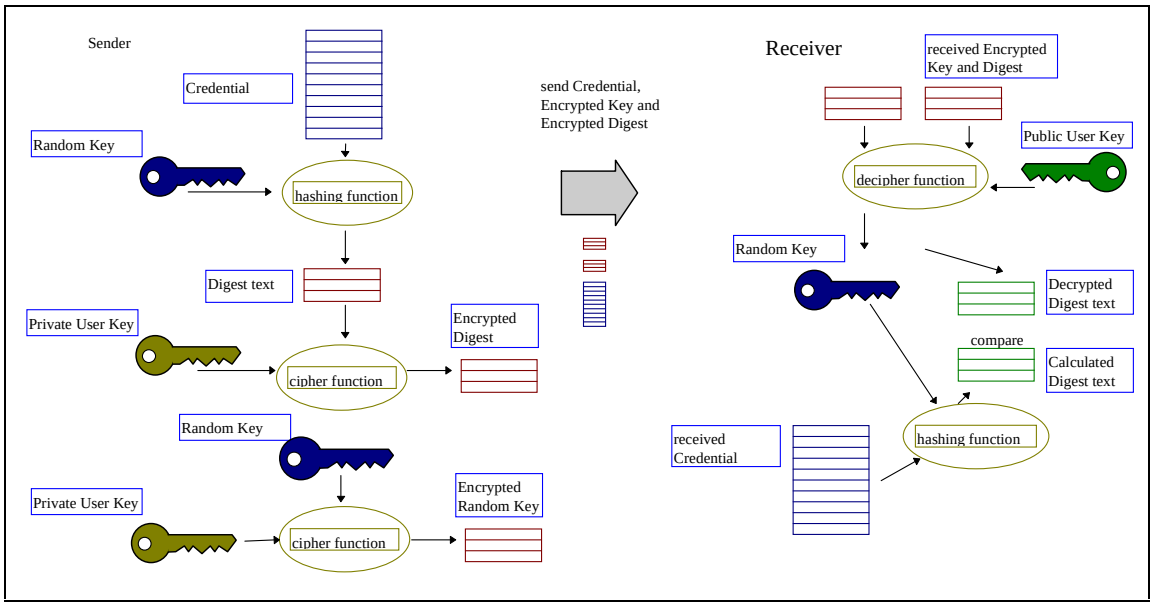

Figure 3-25: CIP B Asymmetric Key Authentication

Note that some public/private key cryptosystems have slightly more simple approach whereby the message digest function does not require an additional key, e.g. PGP. This means that there is no need to generate a random key to calculate the message digest. It would be possible for a Retrieval Manager to agree with its registered users on the use of PGP for authentication.

Certification Authorities

The problem of where to register and obtain public keys can be solved through the use of a "Certification Authority". A certification authority is a highly trusted body which maintains authenticated copies of public keys. The Certification Authority will perform a number of checks when registering a public key against individuals and organisations, these checks are intended to ensure beyond any reasonable doubt that the key and key provider are genuine. The Certification Authority acts as a trusted third party between the two parties engaged in a transaction. The Certification Authority signs each of its transactions using its own private key, therefore authenticating its own messages. The public key of the Certification Authority is distributed as widely as possible to provide maximum benefit to users.

In the CIP context, their use can be explained as follows:

A user is registered with Certification Authority, "TrustMe". The user responds to an **AuthenticationRequest** by signing the provided **credential** and specifying the cryptographic system used together with the Certification Authority. If the Retrieval Manager does not already have the public key of the user, they can request the public key of the user from the Certification Authority. This request (and the registration of the key) are performed over the Certification Authority's trusted communication links and are not part of CIP.

The choice of Certification Authority is not specified by CIP B.

For asymmetric keys, the format of the *encrypted* structure in an *AccessControlRequest* is:

crypt type = CryptTypeName,[Certification Authority]

credential = {Timestamp, UID, Operation, RandPad, Timestamp}

data = NULL

"UserId" is the user identifier against which the user is being authenticated. For pass-through authentication (see Section 3.9.7.8), the user identifier must be preceded by the identifier of the remote Retrieval Manager at which the user is authenticated[48].

"Operation" is the complete CIP operation being authenticated. If the CIP Order ES Request is being authenticated then this ASN.1 structure (and its content as received by the target) is repeated in the credential, so that the original order request details will be returned in the credential. This is to ens ure that the signed operation can be used for non-repudiation purposes. Note that the CIP client must provide some form of notification to the user that a particular *operation* is being signed in order to prevent misuse of this *facility*. "Operation" is required only when non-repudiation is desired (i.e. for an order submission request). Otherwise, "Operation" does not need to be included in the credential.

"RandPad" is a randomly generated sequence of data produced by the *target* to pad out the credential

"Timestamp" is a Retrieval Manager (*target*) generated timestamp.

Note that the CIP client must provide some form of notification to the user that a particular *operation* is being signed in order to prevent misuse of this facility.

The format of the *encrypted* structure for the *AccessControlResponse* is:

crypt type = CryptTypeName,[Certification Authority]

credential = {Timestamp, UID, [Operation], RandPad, Timestamp}

data = $H_{CRYPT\_TYPE}\{ H_{MD5}\{Credential \}\}$

Where

"CryptTypeName" is the name of the cryptographic approach used by the user

"Certification Authority" is an optional field naming the certification authority

"Timestamp" is a Retrieval Manager (*target*) generated timestamp.

"UserId" is the user identifier against which the user is being authenticated preceded, when appropriate by the identifier of the remote Retrieval Manager at which the user is authenticated.

"Operation" is an optional field provided if the authentication request is to authenticate a particular operation for non-repudiation purposes.

"RandPad" is a randomly generated sequence of data produced by the *target* to pad out the credential

$H_{MD5}\{\}$is the result of applying the MD5 hashing algorithm to the content described in the braces.

$H_{CRYPT\ TPYE}\{\}$is the result of applying the specified (by "CryptTypeName") encryption algorithm to the content described in the braces.

3.9.7.4 User Groups

The CIP B adopts an approach to user groups and access rights that is similar to operating systems such as UNIX with some simplifications.

To simplify the management of access rights to data and services, access rights are associated with groups of users rather than individual users. Each user is a member of at least one group, and each group has a defined set of capabilities and order profile. A defined user group can contain zero or more users, this means that it is possible for a Retrieval Manager to create user groups with specific access rights, but that there may or may not be users associated with the group. There is always a "guest" group which is the default group for non-authenticated users.

Group identifiers are not exchanged by Retrieval Managers nor are they assumed to be the same across the CIP domain. Each Retrieval Manager has a set of groups that is defined for their users. Each Retrieval Manager has the standard group "guest" as part of this set. Members of the group must be able to access a consistent set of services across the ICS to allow (at least) the capability of searching collections.

3.9.7.5 Users with more than one Account in the CIP Domain

It is possible that a given user will have more than one account within the CIP domain (i.e. they may be a registered user at more than one Retrieval Manager). In this situation there are significant advantages if the user provides the same public key for all Retrieval Managers. This will enable the CIP client to respond to challenges from CIP Retrieval Managers with the minimum of user intervention. It is also suggested that the user should have a dedicated CIP key pair and that they may maintain this separately from their other work and private keys.

For symmetric key approaches, more than two copies of the user's key will exist. Again, a CIP specific key is strongly recommended. The principle of minimum user intervention can be preserved but multiple copies of the same user identity and key pair mean that it is possible for people with system level access to one of the Retrieval Managers to obtain the user key and use it to access other Retrieval Managers. With symmetric keys and more than one user account, the ICS therefore requires a degree of trust between each Retrieval Manager.

3.9.7.6 User Profile

A user profile consists of information related to the individual user, such as full name, title, telephone number etc. In addition a user may inherit group level information such as the pricing agreement and access rights.

Some of the information in a user profile may be overwritten under user request, other details (such as pricing structure) are fixed and can only be changed through system management of the local Retrieval Manager and User Profile System

In the CIP Domain, Retrieval Managers need to be able to recognise each other in the same way that they recognise and authenticate users. This means that there will be entries in the User Profile System for each Retrieval Manager which correspond to the other Retrieval Managers with whom an operating agreement exists. These entries must be treated in the same way as the standard user entry, i.e. there must be an identifier and contact information and the Retrieval Manager must be associated with a group which provides the appropriate access rights for Retrieval Managers.

Although the CIP does not mandate that a Retrieval Manager uses the CIP for the search and retrieval of user profile information, the CIP provides definitions to support this. User information may be accessed via the CIP using the *search* and *retrieval facilities*. The definition of the *target* user database is not specified by the CIP. However, the CIP provides the definition of the user descriptor that will be retrieved in the CIP Schema (see Appendix C).

3.9.7.7 Authenticated Sessions and Operations

CIP B allows the *origin* and *target* to exchange information at the start of a session (or at any time during the session) which enables the identity of the parties to be confirmed. Exchange of confirmed identities can be used to start a session, where subsequent operations are allowed with the privilege of the identified user. Note that this approach does not guarantee that the session cannot be hijacked by another party following the exchange of authenticated messages. The Retrieval Manager may therefore request authentication of any or all privileged operations in order to increase confidence in the security of the transaction.

Figure 3-26 is a timeline diagram of the associations and operations that may be created during a user session with a local Retrieval Manager and shows where the *AccessControlRequest* and *AccessControlResponse* may be used (shown as authentication request and response respectively).

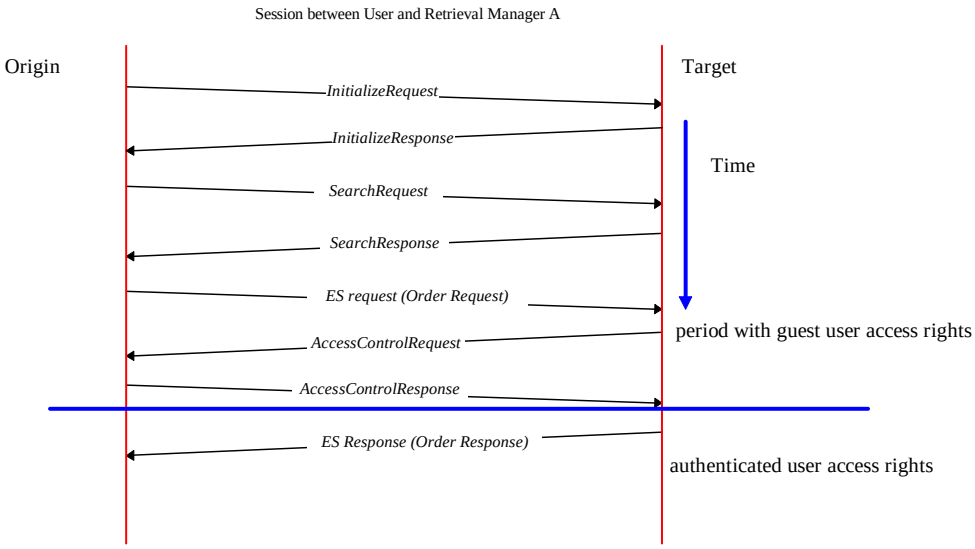

Figure 3-26 Use of Access Control during an Association

3.9.7.8 Distributed Sessions

CIP B allows requests at a local Retrieval Manager to be propagated to remote Retrieval Managers. One of the most important capabilities that this provides is searches across related data supported by different Retrieval Managers.

When a user requests a service which requires a remote operation (i.e. an operation on another Retrieval Manager) to be executed, then the local Retrieval Manager needs to establish an association with the remote Retrieval Manager. To do this the local Retrieval Manager sends an *Initialisation request* to the remote Retrieval Manager, including the user profile information (using the **`UserInformationField`**).

When an operation on the remote Retrieval Manager requires authentication, the remote Retrieval Manager will issue a challenge over the association established between itself and the local Retrieval Manager. Three cases are distinguished:

1) If there is a mutual agreement between the local Retrieval Manager and the remote Retrieval Manager regarding the management or recognition of their users, a one-step pass-through authentication is performed.

2) If there is an agreement between the local Retrieval Manager and the remote Retrieval Manager that the local Retrieval Manage may act as proxy for its users, proxy authentication is performed.

3) Otherwise, a two step pass-through authentication is performed.

Requests which result in operations across more than one Retrieval Manager may not always require authentication. This means that the “guest” level of access may provide the user with access rights which enable searches for information across the CIP domain.

For all distributed access, the local Retrieval Manager is responsible for the distribution of its user requests. When an order is requested from a remote Retrieval Manager by the local Retrieval Manager on behalf of a user, the local Retrieval Manager is responsible for the order (a “proxy order”). It is possible for a non-authenticated user to perform a local or remote order through the CIP. The ability to execute the order will depend on the access rights provided to guest users on the Retrieval Manager(s) and the terms of the agreement between the two Retrieval Managers in the case of a proxy order.

Proxy Authentication

When proxy authentication is in effect, the local Retrieval Manager acts by proxy for the user and therefore substitutes its own **`userId`** for the user's **`userId`** whenever a **`userId`** is included in an operation (such as *Initialize* or CIP Order ES). For operations requiring authentication, the remote Retrieval Manager will authenticate the operation with the passed local Retrieval Manager's **`userId`** via an *AccessControlRequest*. The local Retrieval Manager will then respond to the challenge with an *AccessControlResponse*.

Note that proxy authentication requires a high level of agreement between the two catalogue systems in order to agree what access rights (and billing system) will apply for such remote access requests.

The process of establishing a remote session using proxy authentication is shown in Figure 3-27.

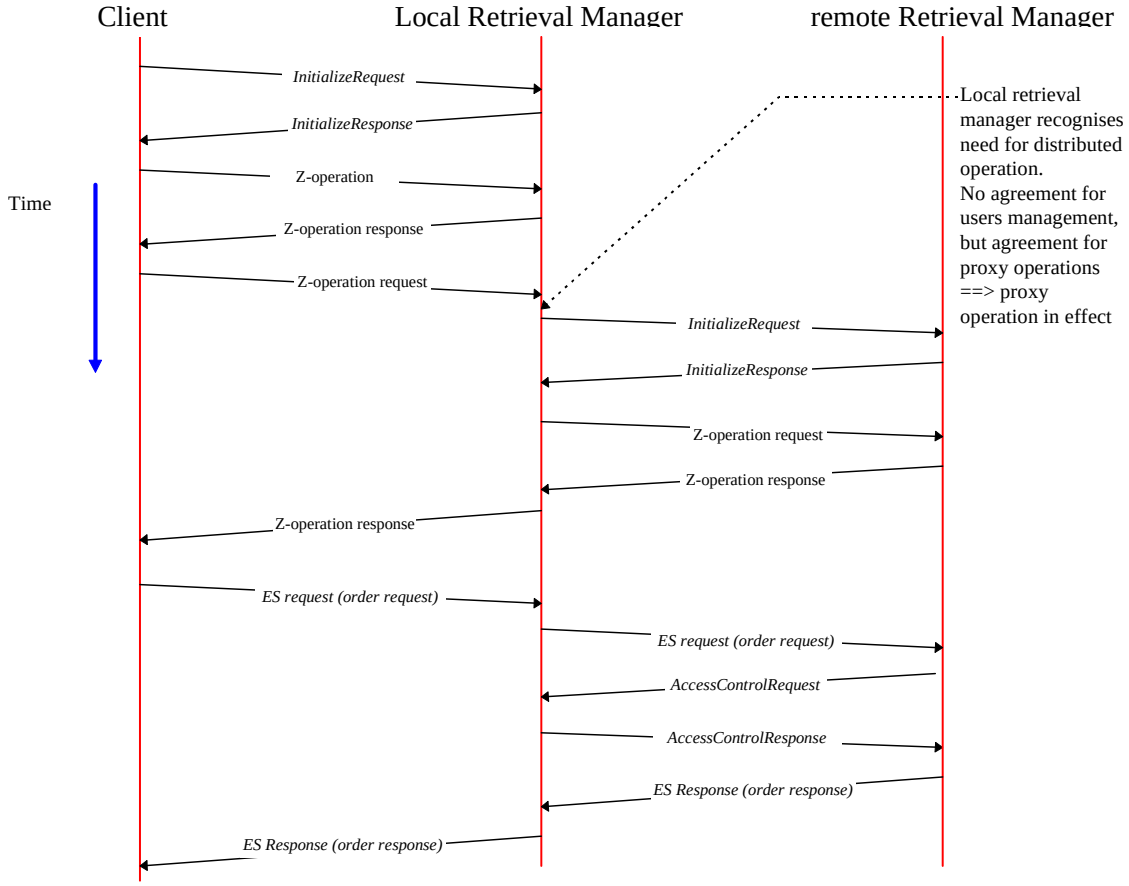

Figure 3-27 Proxy remote sessions

Pass Through Authentication

When pass-through authentication is in effect, the local Retrieval Manager is used to pass operations between the client and the remote Retrieval Manager, without altering the **`userId`** which may be contained in the operations exchanged. For operations requiring authentication, two type of pass-through authentication are distinguished, depending on the knowledge regarding their respective users that the local Retrieval Manager and the remote Retrieval Manager share:

- **one-step pass-through authentication**: if there is a mutual agreement between the local Retrieval Manager and the remote Retrieval Manager regarding the management or recognition of their users, the remote Retrieval Manager is able to identify the user from the user profile information which was sent when the initialisation operation was performed. The

remote Retrieval Manager is therefore able to determine the user's **`userId`** in its own domain (in case it differs from the **`userId`** in the local Retrieval Manager's domain). The authentication of the operation can therefore be performed directly.

- **two-step pass-through authentication**: if the remote Retrieval Manager does not have a mutual agreement with the local Retrieval Manager regarding the management of the users, the remote Retrieval Manager is not able to identify with confidence the user from the user profile information which was sent when the initialisation operation was performed. The remote Retrieval Manager must therefore issue a first *AccessControlRequest* requesting the user to provide his **`userId`** at the remote Retrieval Manager (rather than his **`userId`** at the local Retrieval Manager). This *AccessControlRequest* is sent to the local Retrieval Manager, which simply forwards it to the client. Upon reception of the *AccessControlRequest*, the client provides the user's **`userId`** at the remote Retrieval Manager in the *AccessControlResponse* and sends it the local Retrieval Manager, which forwards it to the remote Retrieval Manager. The remote Retrieval Manager is then able to identify the user and performs the authentication of the operation.

The remote Retrieval Manager can then authenticate the operation with the user's **`userId`** at the remote Retrieval Manager via an *AccessControlRequest*. For this purpose, the user's **`userId`** is prefixed with the identifier of the remote Retrieval Manager so that the client is able to determine that the challenge is requested by the remote Retrieval Manager rather than the local Retrieval Manager. The *AccessControlRequest* is sent to the local Retrieval Manager, which simply forwards it to the client. The client then responds to the challenge with an *AccessControlResponse*, which is turn forwarded by the local Retrieval Manger to the remote Retrieval Manager.

The process of establishing a remote session using the one-step pass through authentication is show in Figure 3-28, whereas the two-step pass-through authentication is shown in Figure 3-29.

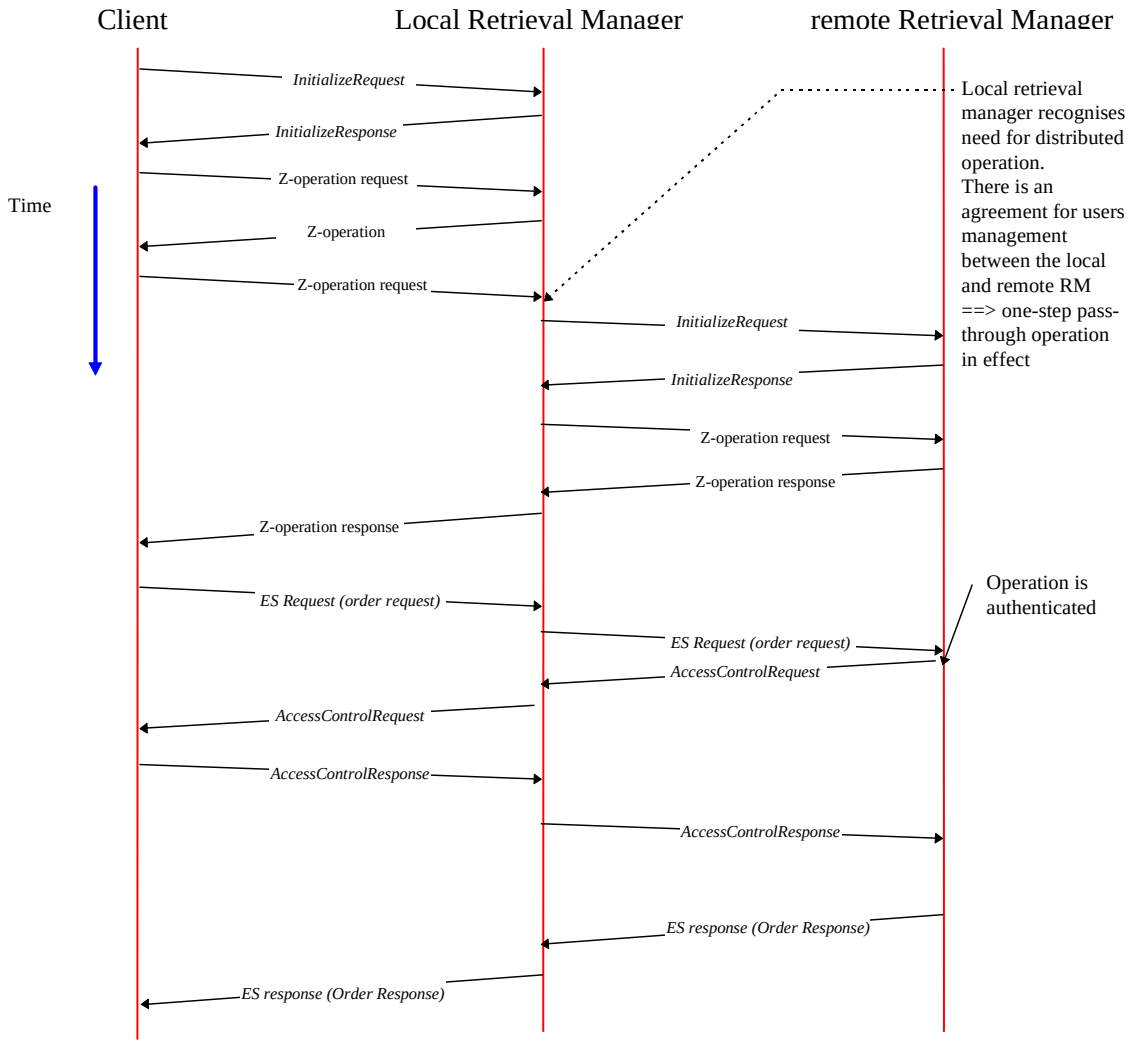

Figure 3-28 One-Step Pass Through Remote Sessions

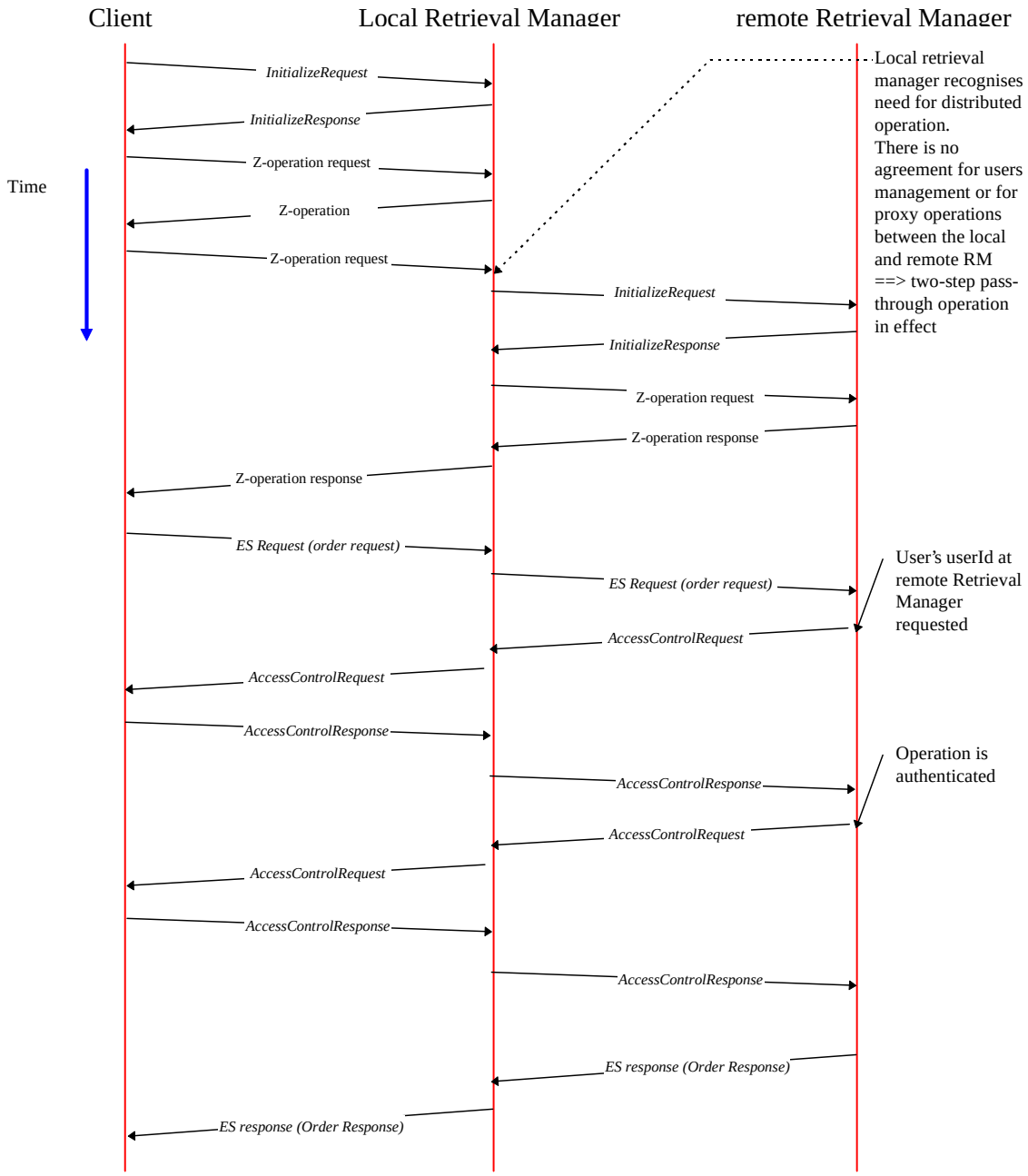

Figure 3-29 Two-Step Pass Through Remote Sessions

3.9.7.9 Message Encryption

CIP B does not specify an approach for the exchange of encrypted messages. This is due to the legal and practical issues surrounding the use of encryption techniques over a co-operative international network